Soft Warehouse

Handcrafted Software for the Mind

User Manual

DERIVE

VERSION 3

A Mathematical Assistant for Your Personal Computer

User Manual

DERIVE®

VERSION 3

A Mathematical Assistant for Your Personal Computer



Printed on recycled paper with soy ink.

Soft Warehouse, Inc., 3660 Waialae Avenue, Suite 304
Honolulu, Hawaii, U.S.A. 96816-3236

Information in this document is subject to change without notice and does not represent a commitment on the part of **Soft Warehouse, Inc**. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy this software on magnetic tape, disk, or any other medium, for any purpose other than the purchaser's personal use.

Copyright © 1988 through 1996 by Soft Warehouse, Inc.
All rights reserved worldwide

No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the express written permission of **Soft Warehouse, Inc**.

If you have comments about or difficulties with this documentation or the software, please complete the *DERIVE Problem Report Form* at the back of this manual and send it to **Soft Warehouse, Inc.** Problem reports can also be submitted on the **Soft Warehouse, Inc.** web page at http://www.derive.com.

First edition: September 1988
Second edition: February 1989
Third edition: June 1989
Fourth edition: August 1990
Fifth edition: December 1992
Sixth edition: December 1993
Seventh edition, first printing: September 1994
Seventh edition, second printing: January 1995
Seventh edition, third printing: June 1995
Seventh edition, fourth printing: March 1996

DERIVE is a registered trademark of Soft Warehouse, Inc.
muLISP is a trademark of Soft Warehouse, Inc.
AcroSpin is a trademark of Acrobits.
AT&T is a registered trademark of American Telephone and Telegraph Corporation.
Hercules is a trademark of Hercules Computer Technology, Inc.
Hewlett-Packard, HP 95LX, HP 100LX, and **HP 200LX** are trademarks of Hewlett-Packard.
IBM is a registered trademark of International Business Machines Corporation.
Intel and **Pentium** are registered trademarks of Intel Corporation.
Microsoft, MS-DOS, and **Windows** are registered trademarks of Microsoft Corporation.
NEC is a registered trademark of Nippon Electric Company.
T3100 and **T5100** are trademarks of Toshiba Inc.

Acknowledgments

Making mathematics more exciting and enjoyable is the driving force behind the development of the ***DERIVE*** program. The system is designed to eliminate the drudgery of performing long tedious mathematical calculations. This gives you the freedom to explore different approaches to problems -- approaches that you probably would not even consider if you had to do the calculations by hand.

To accomplish this ***DERIVE*** not only has to be a tireless, powerful, and knowledgeable mathematical assistant, it must be an easy, natural, and convenient tool.

The following is a partial list of the many people that helped us achieve these ambitious and sometimes conflicting goals: Chris Arney, Matias Arnold, Klaus Aspetsberger, Gunnar Bäckström, John Berry, Ernst Boeck, Josef Böhm, Don Cohen, Joel Cohen, Martin Cohen, Robert Corless, Tom Cromer, Paul D'Carpio, J.S. Dowker, Paul Drijvers, Axel Dubois, Hans Dudler, Wade Ellis, Terence Etchells, Wuchen Fei, James FitzSimons, David Flesner, Ralph Freese, Akio Furukawa, Robert Gilbert, Larry Gilligan, Jerry Glynn, Constantino Grosse, Leon Hall, Martti Hamina, Reif Hammond, Donald Hazlewood, Alan Head, Michael Henle, Fred Hoffman, Marko Horbatsch, John Hosack, Norman Hughes, Alice Ihori, Patricia Ihori, Ron Ihori, Adi Ben-Israel, David Jeffrey, Jerry Johnson, W. Kahan, Anders Kallstrom, Peter Kaomea, Hans-Ake Karlsson, Wolfram Koepf, Bernhard Kutzler, Eric Lear, Joel Lehmann, Carl Leinbach, Patti and Arnold Littlefield, James Marquardt, Takafumi Matsumoto, Robert Moore, Duncan Murdoch, Paul Nielan, John Ogilvie, Robert Orban, Warren Page, Norman Peterson, Thomas Phipps, Andre Ratel, Gert Renkema, John Saeger, Peter Schofield, Steven Schonefeld, Robert Setif, Ray Shaw, David Sjöstrand, Thomas Speziale, David Stegenga, Karen Stoutemyer, Don Taylor, Doug Telford, David Terpenning, Frederick Tydeman, Rolf Wasen, Hidetoshi Watanabe, Greg Whitten, Walter Wild, and Georg Ziemes.

Mahalo nui loa (that's Hawaiian for thank you very much) to everyone who helped.

Albert Rich
Joan Rich
Theresa Shelby
David Stoutemyer

Table of Contents

Title Page i
Copyright Notice ii
Acknowledgments iii
Table of Contents iv

1. Introduction

1.1 System Contents 3
1.2 System Requirements 6
1.3 Making A Working Copy 8
1.4 Installing on a Local Area Network 11
1.5 Controlling ***DERIVE XM*** Memory Usage 14
1.6 Key Notation and Entry 15

2. Fundamentals

2.1 Starting ***DERIVE*** 17
2.2 The ***DERIVE*** Screen 18
2.3 Using The Menu System 20
2.4 Line Editing 22
2.5 Getting Help 24
2.6 Quitting ***DERIVE*** 25
2.7 Executing DOS Commands 26
2.8 Saving Expressions 27
2.9 Loading Expressions 28
2.10 Printing Expressions 30
2.11 Printing Screen Images 32
2.12 Display Modes and Colors 34
2.13 Window Commands 38
2.14 Saving Current State 40

3. Arithmetic

3.1 Entering Expressions 43
3.2 Simplifying Expressions 47
3.3 Highlighting Expressions 48
3.4 Entering Functions 52
3.5 Building Expressions 54
3.6 Moving Expressions 56
3.7 Removing, Unremoving, and Renumbering Expressions 57
3.8 Controlling Precision 59
3.9 Controlling Notation 64
3.10 Approximating Irrational Numbers 66
3.11 Controlling Radix Base 67
3.12 Annotating Expressions 69
3.13 Controlling Expression Format 70

4. Algebra

4.1 Entering Variables 71
4.2 Simplifying Expressions 74
4.3 Ordering Variables 78
4.4 Expanding Polynomials 81
4.5 Choosing Complex Branches 84
4.6 Factoring Expressions 87
4.7 Partial Fraction Expansion 92
4.8 Substituting Values 94
4.9 Entering Infinite Values 98
4.10 Declaring Domains Of Variables 99
4.11 Assigning Values to Variables 103
4.12 Defining Functions 106
4.13 Entering Equations and Relations 108
4.14 Solving Equations and Relations 112
4.15 Solving Systems of Equations 115
4.16 Boolean Operators and Truth Tables 119

5. Plotting

5.1 2D-plot Window 121
5.1.1 Plotting Univariate Expressions 122
5.1.2 Setting 2D-plot Window Preferences 127
5.1.3 Framing 2D Plots 129
5.1.4 Multiple Plots 137
5.1.5 Plot Speed and Accuracy 139
5.1.6 Polar Plots 142
5.1.7 Parametric Plots 145
5.1.8 Data Matrix Plots 147
5.1.9 Implicit Plots 149
5.2 3D-plot Window 153
5.2.1 Plotting Bivariate Expressions 153
5.2.2 Framing 3D Plots 157
5.2.3 Rotating 3D Plots 159
5.2.4 Animation with **AcroSpin** 160

6. Functions and Constants

6.1 Exponential Functions 164
6.2 Logarithmic Functions 166
6.3 Trigonometric Functions 167
6.4 Inverse Trigonometric Functions 171
6.5 Hyperbolic Functions 173
6.6 Inverse Hyperbolic Functions 174
6.7 Piecewise Continuous Functions 175
6.8 Complex Variable Functions 177
6.9 Probability Functions 178
6.10 Statistical Functions 180
6.11 Error and Zeta Functions 183
6.12 Financial Functions 184
6.13 Number Theory Functions 186
6.14 Expression Decomposition Functions 187

7. Calculus

7.1 Limits 189
7.2 Differentiation 192
7.3 Taylor Polynomial Approximations 194
7.4 Integration 196
7.5 Sums 201
7.6 Products 203

8. Vectors and Matrices

8.1 Entering Vectors and Matrices 205
8.2 Generating Vectors, Matrices, and Tables 207
8.3 Vector Manipulation Functions 209
8.4 Vector Operations 212
8.5 Matrix Operations 215
8.6 Row Echelon Form 218
8.7 Eigenvalues 220
8.8 Vector and Matrix Algebra 221
8.9 Differential Vector Calculus 222
8.10 Integral Vector Calculus 225

9. Utility Files

9.1 SOLVE.MTH — Solving Nonlinear Systems & Complex Solutions . 228
9.2 VECTOR.MTH — Additional Vector and Matrix Functions 232
9.3 NUMERIC.MTH — Numerical Differentiation and Integration 240
9.4 DIF_APPS.MTH — Applications of Differentiation 242
9.5 INT_APPS.MTH — Applications of Integration 247
9.6 ODE1.MTH — First-order Ordinary Differential Equations 252
9.7 ODE2.MTH — Second-order Ordinary Differential Equations 259
9.8 ODE_APPR.MTH — Approximate Solution of ODEs 263
9.9 RECUREQN.MTH — Recurrence Equations 269
9.10 APPROX.MTH — Pade Rational Approximation 272
9.11 EXP_INT.MTH — Exponential, Log, Sine, and Cosine Integrals . . . 273
9.12 PROBABIL.MTH — Probability Functions 275
9.13 FRESNEL.MTH — Fresnel Integrals 278
9.14 BESSEL.MTH — Bessel and Airy Functions 279

9.15 HYPERGEO.MTH — Hypergeometric Functions 281
9.16 ELLIPTIC.MTH — Elliptic Integrals 282
9.17 ORTH_POL.MTH — Orthogonal Polynomials 283
9.18 ZETA.MTH — Generalized Riemann Zeta Functions 286
9.19 GRAPHICS.MTH — Plotting Space Curves & Complex Expressions 287
9.20 NUMBER.MTH — Number Theory Functions 295
9.21 MISC.MTH — Miscellaneous Utility Functions 297

10. Programming

10.1 The ITERATES Function 303
10.2 The ITERATE Function 307
10.3 IF Expressions 309
10.4 Nested IF Functions 312
10.5 The Logical Operators 313
10.6 Recursive Functions 317
10.7 Mutual Recursion 320
10.8 Epilogue 322

Appendixes

Questions and Answers 323
Screen Capture Programs 329
Additional Resources 331
Menu Commands 341
Line Editing Commands 347
Highlighting Commands 349
Function Key Commands 350
Index 351
Problem Report Form 369
License Agreement 371
Limited 90-Day Warranty 373

Chapter 1. Introduction

Congratulations on your purchase of the ***DERIVE*** and/or ***DERIVE XM*** mathematical assistant programs. These powerful, yet easy to use, expert systems incorporate the knowledge required to simplify, solve, and plot a wide range of mathematical equations and expressions, both symbolically and numerically.

The operation of both programs is virtually identical. The primary difference lies in the amount of memory each can take advantage of and hence how large a problem each can handle. Regular ***DERIVE*** takes advantage of up to 640 kilobytes of memory and can handle small to medium size math problems. ***DERIVE XM*** takes advantage of up to 4 gigabytes (4 billion bytes) of extended memory and can handle huge problems.

Before Doing Anything Else: Register Today!

Please fill out and send in the Registration Card that comes in the sealed envelope with the ***DERIVE*** or ***DERIVE XM*** diskette. No postage is required if mailed within the United States. Otherwise the Registration Card should be mailed in a stamped envelope. You need to register to receive technical support. Also we need your address to send you information about new versions and upgrade discount offers.

This user manual describes the operation of both systems while noting the few differences between them. The information in the manual is arranged as follows:

- **Chapter 1. Introduction** summarizes the major features of ***DERIVE***, the files on the ***DERIVE*** diskette, the necessary computer requirements, the procedure for copying the ***DERIVE*** diskette, and the notation used for keys on the keyboard.
- **Chapter 2. Fundamentals** describes how to launch and quit ***DERIVE***; how to use the menu system; how to load, save, and print the mathematical expressions that you create; how to change the display modes; and how to open and close windows.
- **Chapter 3. Arithmetic** describes how to enter, simplify, and select numeric expressions; how to build, move, and remove expressions; and how to set the precision, notation, and radix base for numbers.
- **Chapter 4. Algebra** describes how to enter, simplify, expand, and factor algebraic expressions; how to substitute in expressions; how to declare the domains of variables; and how to enter and solve equations.

- **Chapter 5. Plotting** describes how to make 2D explicit, implicit, and parametric plots in rectangular or polar coordinates; and how to make 3D wire-frame plots with or without hidden line removal.
- **Chapter 6. Functions and Constants** describes the pre-defined mathematical functions and constants that ***DERIVE*** knows how to simplify.
- **Chapter 7. Calculus** describes how to find limits, derivatives, Taylor approximations, integrals, and sums or products in closed form.
- **Chapter 8. Vectors and Matrices** describes how to enter and manipulate vectors and matrices using the pre-defined functions and operators provided by ***DERIVE***.
- **Chapter 9. Utility Files** describes how to use the facilities provided by the utility files distributed with ***DERIVE***.
- **Chapter 10. Programming** describes how to define functions using iteration, recursion, and conditionals.

Appendixes

- **Questions and Answers** answers some of the questions commonly asked by new users of ***DERIVE***.
- **Screen Capture Programs** lists several popular programs for sending graphics screen images to printers or for saving them in files.
- **Additional Resources** provides information on the ***DERIVE*** User Group, the *International* ***DERIVE*** *Journal*, the ***DERIVE*** Internet mailing list, the ***DERIVE*** Electronic Bulletin Board, and a list of books relating to ***DERIVE***.
- **Menu Commands** lists and briefly summarizes the function of each menu command in the three ***DERIVE*** windows (Algebra, 2D-plot, and 3D-plot).
- **Line Editing Commands** lists and briefly summarizes the commands for line editing and for entering Greek and math symbols.
- **Highlighting Commands** lists and briefly summarizes the commands for highlighting expressions and subexpressions.
- **Function Key Commands** lists and briefly summarizes all the function key commands in ***DERIVE*** including those for switching windows, line editing, switching display modes, zooming plot scales in and out, and printing and saving screen images.
- **Index** is a comprehensive index including all ***DERIVE***'s functions, operators, menu commands, and line editing commands.

- **Problem Report Form** is a form for reporting bugs, deficiencies, or documentations errors in ***DERIVE***.
- **End-User License Agreement and Limited Warranty** is your license to use the ***DERIVE*** software and our limited warranty for the software.

1.1 System Contents

DERIVE consists of the ***DERIVE* User Manual** and the following files on the ***DERIVE* diskette** (additional files may be included):

- README describes new features not yet included in this manual. To display the contents of this file enter the command TYPE README at the DOS prompt.

Program files:

- DERIVE.EXE is the program execution file for ***DERIVE*** (see Section 2.1).
- DERIVE.SYS is the system file for the extended memory version of ***DERIVE***. This file must reside in the same drive and directory as DERIVE.EXE. It is only distributed with ***DERIVE XM***.
- EDPMI.OVL is the Ergo DPMI overlay program that enables ***DERIVE XM*** to use extended memory. It is only distributed with ***DERIVE XM***.
- DERIVE.PIF is the Microsoft Windows PIF file for the extended memory version of ***DERIVE***. It is only distributed with ***DERIVE XM***.
- DERIVE.HLP is the help file for ***DERIVE*** (see Section 2.5).
- HERCULES.COM is a resident program for running ***DERIVE*** on computers with Hercules-compatible graphics cards (see Section 2.12).

Demonstration files: (to run, see Section 2.9)

- ARITH.DMO is the arithmetic demonstration file (see Chapter 3).
- ALGEBRA.DMO is the algebra demonstration file (see Chapter 4).
- TRIG.DMO is the trigonometry demonstration file (see Chapter 6).
- FUNCTION.DMO is the exponential, logarithm, probability, and statistical demonstration file (see Chapter 6).
- CALCULUS.DMO is the calculus demonstration file (see Chapter 7).

- MATRIX.DMO is the vector and matrix demonstration file (see Chapter 8).

Plot example files:

- PLOT2D.MTH is the 2-dimensional plot example file (see Section 5.1.1).
- PLOTPARA.MTH is the polar and parametric plot example file (see Section 5.1.6 and 5.1.7).
- PLOT3D.MTH is the 3-dimensional plot example file (see Section 5.2.1).

Utility files:

- ENGLISH.MTH defines unit conversion factors for the English unit system used in the United States (see Section 4.11).
- METRIC.MTH defines unit conversion factors for the metric unit system (see Section 4.11).
- PHYSICAL.MTH defines fundamental physical constants to high precision (see Section 4.11).
- SOLVE.MTH defines functions for finding approximate real or complex roots of nonlinear systems of simultaneous algebraic equations (see Section 9.1).
- VECTOR.MTH defines functions that automate some of the operations commonly performed on vectors and matrices (see Section 9.2). It supplements those pre-defined vector and matrix functions described in Chapter 8.
- NUMERIC.MTH defines functions for numeric differentiation and integration (see Section 9.3).
- DIF_APPS.MTH defines functions that use symbolic differentiation (see Section 9.4).
- INT_APPS.MTH defines functions that use symbolic integration (see of Section 9.5).
- ODE1.MTH defines functions for solving first order ordinary differential equations (see Section 9.6).
- ODE2.MTH defines functions for solving second order ordinary differential equations (see Section 9.7).
- ODE_APPR.MTH defines functions for finding approximate solutions of ordinary differential equations (see Section 9.8).

- RECUREQN.MTH defines functions for solving recurrence equations (see Section 9.9).
- APPROX.MTH defines a function for finding Pade rational approximations (see Section 9.10).
- EXP_INT.MTH defines functions for computing exponential, log, sine, and cosine integrals (see Section 9.11).
- PROBABIL.MTH defines the incomplete gamma function and its relatives (see Section 9.12). It supplements those pre-defined probability functions described in Section 6.9.
- FRESNEL.MTH defines functions for computing Fresnel integrals (see Section 9.13).
- BESSEL.MTH defines functions for computing Bessel and Airy functions (see Section 9.14).
- HYPERGEO.MTH defines functions for computing hypergeometric functions (see Section 9.15).
- ELLIPTIC.MTH defines functions for computing elliptic integrals (see Section 9.16).
- ORTH_POL.MTH defines functions for computing orthogonal polynomials (see Section 9.17).
- ZETA.MTH defines functions for approximating functions related to the Riemann zeta function (see Section 9.18).
- GRAPHICS.MTH defines functions that aid in plotting space curves, parametric surfaces, and complex-valued expressions (see Section 9.19).
- NUMBER.MTH defines functions useful for investigations in number theory (see Section 9.20).
- MISC.MTH defines miscellaneous utility functions (see Section 9.21).
- SUPPORT.PAS defines functions in Pascal for use with the PAS files generated by ***DERIVE*** (see Section 2.8).

User Contributed files:

The USERS directory on the ***DERIVE*** diskette includes utility packages contributed at no charge by various ***DERIVE*** users. The README file includes a list of the packages available.

Some of the MTH files for these packages are very large. To save space when loading them, use a **Transfer Load Utility** command rather than a **Transfer Load Derive** command (see Section 2.9).

Corresponding to each MTH file is a DOC file that documents the package. Some packages also have a DMO file that can be run after the MTH file is loaded. If you have questions about a package, contact the package's author directly. The author's name is given in the package's DOC file.

1.2 System Requirements

This section summarizes the minimum computer hardware and disk operating system (DOS) software required to run ***DERIVE*** and ***DERIVE XM***.

DERIVE for PC compatible computers:

- Any PC or AT compatible computer based on the Intel 8086 or compatible microprocessor running MS-DOS version 2.1 or later. MS-DOS compatible operating systems include Microsoft Windows, Microsoft Windows NT, IBM PC-DOS, IBM PS/2, and IBM OS/2.
- A monochrome display adapter (MDA) and a compatible monitor; or a graphics display adapter (CGA, EGA, MCGA, VGA, or Hercules) and a compatible monitor.
- At least 512 kilobytes of conventional computer memory (***DERIVE*** can take advantage of up to 640 kilobytes of conventional memory).
- One double-sided (720 kilobyte) 3½ inch diskette drive.

DERIVE can also be downloaded from a desktop computer and run on an HP 95LX, 100LX, or 200LX Palmtop computer that has at least 1 megabyte of built-in RAM. Also recommended is a PCMCIA RAM card on which to store the ***DERIVE*** program files. ***DERIVE*** supports the 40 column, 240 by 128 graphics mode of the 95LX and the 80 column, 640 by 200 (i.e. CGA) graphics mode of the 100LX and 200LX. When running on these computers, ***DERIVE*** automatically takes measures to minimize battery usage.

DERIVE for PC-9801 (Japanese) compatible computers:

- An NEC PC-9801 Personal Computer *or* a PC-9801 compatible computer running MS-DOS version 2.1 or later, or compatible operating system software.
- A monochrome monitor or a color monitor.

- At least 512 kilobytes of computer memory. ***DERIVE*** can take advantage of up to 640 kilobytes of memory.
- At least one double-sided (1.2 megabyte) 5¼ inch diskette drive *or* one double-sided (1.2 megabyte) 3½ inch diskette drive.

DERIVE XM for 386 PC compatible computers:

- A PC compatible computer based on the Intel 386, 486, Pentium®, or compatible microprocessor running MS-DOS version 3.0 or later. MS-DOS compatible operating systems include Microsoft Windows, Microsoft Windows NT, and IBM OS/2.
- A monochrome display adapter (MDA) and a compatible monitor; or a graphics display adapter (CGA, EGA, MCGA, or VGA) and a compatible monitor.
- At least 2 megabytes of extended memory (***DERIVE XM*** can take advantage of up to 4 gigabytes of extended memory).
- One double-sided, high-density (1.4 megabyte) 3½ inch diskette drive.

1.3 Making a Working Copy

As a convenience to authorized users, the ***DERIVE*** diskette is *not* copy protected. It can be copied using standard DOS copying commands. In return, you are asked to abide by the copyright and *not* distribute copies of the software to unauthorized users.

As a safeguard against the loss, damage, or erasure of the ***DERIVE*** diskette, you should make a working copy of the diskette *before* using ***DERIVE***. (This precaution is *not* necessary for the ROM card version of ***DERIVE***.) If the working copy is made on a diskette, the diskette must have a label bearing the following copyright notice:

***DERIVE*®**
Copyright © 1988 through 1996 by Soft Warehouse, Inc.
All rights reserved worldwide

In addition to the copyright notice, it is a good idea to put the version number of ***DERIVE*** on the diskette label. The version number is on the master ***DERIVE*** diskette and is displayed when ***DERIVE*** starts. When reporting problems with ***DERIVE***, be sure to include the version number on the ***DERIVE* Problem Report Form** printed at the end of this manual.

It is best to write-protect the ***DERIVE*** diskette before making a working copy. A 3½ inch diskette is write-protected by moving the write-protect slide so the window is open. A 5¼ inch diskette is write-protected by covering the notch on the right side of the diskette with a write-protect tab. See your computer manuals for details.

1.3.1 Hard Disk Procedure

If your computer has a floppy diskette drive for drive A and a hard disk drive for drive C, you can make a working copy of ***DERIVE*** on the hard disk as follows:

- Turn on and boot the computer in the ordinary way from either the hard disk or a DOS system diskette in drive A.
- Insert the ***DERIVE*** diskette in drive A.
- Switch to drive C by entering the DOS command

```
C:
```

- Make a new subdirectory named DERIVE (if one does not already exist) by entering the DOS command

```
MD DERIVE
```

- Change to the DERIVE subdirectory by entering the DOS command

```
CD DERIVE
```

- Copy the files from the ***DERIVE*** diskette to the DERIVE subdirectory by entering the DOS command

```
XCOPY A:*.* C:/S
```

- After all the files have been transferred to the hard disk and the DOS prompt is redisplayed, remove the ***DERIVE*** diskette from drive A and store it in a safe, cool, and dry place that is physically separated from the computer.

1.3.2 Diskette Procedure

If your computer has one or two diskette drives but no hard disk drive, you can make a working copy of ***DERIVE*** on a diskette using the DISKCOPY command described in your DOS manual.

Alternatively, if your computer has diskette drives for drives A and B, you can make a working copy of ***DERIVE*** as follows:

- Turn on and boot the computer in the ordinary way from a DOS system diskette in drive A.
- Insert a blank diskette in drive B.
- Format the blank diskette by entering the DOS command

```
FORMAT B:
```

  Alternatively, format the diskette and make it a bootable system diskette by entering the command

```
FORMAT B:/S
```

  Since space is required to store the DOS system files, system diskettes have less space available for user files. Thus, more diskettes may be required to store all the ***DERIVE*** files.

- Insert the ***DERIVE*** diskette in drive A.
- Copy the files from the ***DERIVE*** diskette to the blank working diskette by entering the DOS command

```
XCOPY A:*.* B:/S
```

- After all the files have been transferred to the working diskette and the DOS prompt is redisplayed, remove the ***DERIVE*** diskette from drive A and store it in a safe, cool, and dry place that is physically separated from the working copy of ***DERIVE***.
- Remove the working diskette from drive B and place the previously described copyright notice on it.

1.4 Installing on a Local Area Network

This section applies only to users who have licensed ***DERIVE*** or ***DERIVE XM*** for use on a Local Area Network (LAN). If you wish to obtain a LAN license, please contact your ***DERIVE*** dealer or Soft Warehouse, Inc.

The compact ***DERIVE*** and ***DERIVE XM*** executable files are small by today's standards. Also neither program uses overlay files while running (i.e. once loaded, they make no further access to the hard or floppy disk). Both these facts make installing and running ***DERIVE*** and ***DERIVE XM*** on a LAN relatively simple and straight-forward.

The following describes how to install ***DERIVE*** on a LAN. The procedure for installing ***DERIVE XM*** is the same, except where noted:

- Login with system supervisor privileges and create a directory named DERIVE on the LAN file server. If installing ***DERIVE XM***, name the directory DERIVEXM to avoid a possible name conflict with an existing DERIVE directory.
- Give ***DERIVE*** users "read only" rights to this directory and give the system supervisor all rights (see your LAN manual for details). Note that it is not necessary to establish a command file search path to this directory.
- Using the MS-DOS XCOPY command, transfer all the files on the ***DERIVE*** or ***DERIVE XM*** distribution diskette to the new directory.

> The ***DERIVE*** LAN License requires that reasonable precautions be taken to prevent network users from making copies of the software on their terminals.

On a Novell Netware LAN, to protect ***DERIVE*** and ***DERIVE XM*** from being downloaded, use the Novell FILER utility to designate the file DERIVE.EXE to be "Execute Only" as follows: From the DERIVE directory run the FILER utility and select the "Directory Contents" option. Use the arrow keys to select the DERIVE.EXE file and then select the "View/Set File Information" option. Next select the "Attributes" option and press the **Ins** (Insert) key for a list of available attributes. Finally select the "Execute Only" attribute and then press the **Esc** key to get out of the FILER utility.

DERIVE users often need to save mathematical expressions and/or graphics screen images in files. Normally these files should not be stored in the ***DERIVE*** directory on the file server. Instead they should be saved in a working directory on a diskette, in a directory on the user's own computer, or in the user's own directory on the file server. The DOS drive and directory current at the time ***DERIVE*** is started is the

default working directory. Therefore, ***DERIVE*** should be started from the desired working directory and *not* from the DERIVE directory on the file server.

On a LAN, it is usually preferable to start ***DERIVE*** using a DOS batch file. In addition to starting ***DERIVE***, the batch file can automatically switch to the desired working directory and/or redirect printer output to the desired network printer. The batch file should be named DERIVE.BAT and saved on the file server in the standard batch file directory or any directory to which a command file search path has been established.

For example, if the file server is drive F, the batch file commands

```
@ECHO OFF
F:\DERIVE\DERIVE.EXE %1 %2 %3
```

starts ***DERIVE*** using the current DOS drive and directory as the working directory. If installing ***DERIVE XM***, change the last line of the batch file to read

```
F:\DERIVEXM\DERIVE.EXE %1 %2 %3
```

Alternatively, the batch file commands

```
@ECHO OFF
C:
CD C:\DERIVE
F:\DERIVE\DERIVE.EXE %1 %2 %3
```

makes the DERIVE directory on drive C the working directory. Finally, the batch file commands

```
@ECHO OFF
F:
CD F:\WORKING
F:\DERIVE\DERIVE.EXE %1 %2 %3
```

makes the WORKING directory on the file server the working directory.

If you want to redirect ***DERIVE*** printer output to a network printer, you can include the appropriate printer redirection command before the DERIVE.EXE line in DERIVE.BAT (see your LAN manuals for details). For example, if you are running a Novell LAN and the Netware program CAPTURE.EXE is in the PUBLIC directory on drive F, the command line

```
F:\PUBLIC\CAPTURE L=1 Q=name NT NFF TI=5
```

redirects ***DERIVE*** printer output from port LPT1 to the network printer named *name*. Also tabs are not expanded, extra form feed characters are not sent at the end of print jobs, and data is sent to the printer after a 5 second time-out.

By default, ***DERIVE*** sends printer output to port LPT1. If network printers are assigned to ports LPT2 or LPT3, ***DERIVE*** printer output can sent to these ports by changing the printer port assignment in the ***DERIVE*** initialization file (see Section 2.14).

If the ***DERIVE*** user does not specify a drive and/or directory when attempting to load a utility file, ***DERIVE*** searches for the file in

- the DOS drive and directory current when ***DERIVE*** was started;
- the \DERIVE or \DERIVEXM directory on the current drive; and then
- the directory from which ***DERIVE*** was launched.

Thus, to provide access for all users to a utility file, copy the file to the DERIVE directory on the file server. Then the file will be found by the last search path above.

The directories listed above are also searched to find the DERIVE.INI initialization file when ***DERIVE*** is started (see Section 2.14), and to find the DERIVE.HLP help file when a request is made for help (see Section 2.5).

You may want the state of ***DERIVE*** to be the same for all users when it starts. If this is other than the default initial state, save the desired state in a DERIVE.INI initialization file in the DERIVE directory on the file server (see Section 2.14). Then if individual users wish to modify the initial state, they can save their own DERIVE.INI files in their own working directory.

1.5 Controlling *DERIVE XM* Memory Usage

DERIVE XM normally takes all available extended memory when it starts running in order to maximize the size of math problems it can handle. However, taking all such memory may be undesirable in multi-tasking environments like Microsoft's Windows or IBM's OS/2.

If you have difficulties running ***DERIVE XM***, you can limit the amount of extended memory it uses by issuing an MS-DOS command of the form

```
SET DXM=-MAXP n
```

where *n* is the maximum number of bytes ***DERIVE XM*** is to use. *n* must be an integer entered in decimal notation, or in hexadecimal notation followed by the letter H (e.g. 100000H). Note that ***DERIVE XM*** requires an absolute minimum of 1 megabyte = 100000H bytes of extended memory to run.

Often it is convenient to run another application program from within ***DERIVE XM*** using the **Options Execute** command (see Section 2.7). For this reason ***DERIVE XM*** normally uses only a small amount of conventional memory (i.e. the first megabyte of computer memory) for itself. This leaves the rest of conventional memory available for use by other application programs.

However if your math problems are large and you want to use every bit of available memory, you can force ***DERIVE XM*** to use all of conventional as well as extended memory by issuing the MS-DOS command

```
SET DXM=-MAXR 0
```

In the unlikely event that you want to both limit the use of extended memory *and* use conventional memory, issue the command

```
SET DXM=-MAXP n -MAXR 0
```

> SET DXM commands must be issued *before* launching ***DERIVE XM***. Also, when running ***DERIVE XM*** under Windows or OS/2, these commands must be issued from the MS-DOS prompt *before* launching Windows or OS/2.

Thus, we recommend including SET DXM commands in your AUTOEXEC.BAT file so they are automatically executed each time your computer is turned on.

1.6 Key Notation and Entry

Some keys on computer keyboards may be marked with descriptive symbols instead of names that are spelled out. Throughout this manual these keys are referenced by names printed in bold letters. The following is an alphabetical list of the names and the corresponding symbols used on many keyboards:

Manual Name	Keyboard Symbol	Manual Name	Keyboard Symbol
Alt	Alt	**Ins**	Ins
Backspace	←	**Left arrow**	←
Ctrl	Ctrl	**PgDn**	PgDn
Del	Del	**PgUp**	PgUp
Down arrow	↓	**Right arrow**	→
End	End	**Shift**	⇑
Enter	↵	**Space**	<space bar>
Esc	Esc	**Tab**	⇒
Home	Home	**Up arrow**	↑

Sometimes you need to press two keys at the same time to obtain the desired effect. For example, when the manual says **Shift-Tab**, hold down the **Shift** key and press the **Tab** key. Similarly, **Ctrl-H** means hold down the **Ctrl** key and press the letter **H**, whereas **Alt-E** means hold down the **Alt** key and press the letter **E**.

When the manual says to "enter a command", it means to type the command and then press the **Enter** key.

Chapter 2. Fundamentals

This chapter explains the fundamentals of how to start ***DERIVE***, how to use its menu system, how to save and print mathematical expressions, and how to quit ***DERIVE***. It is essential to understand the information in the first six sections of this Chapter to use the system effectively. The information in the remaining sections need only be briefly scanned until it is actually required.

2.1 Starting *DERIVE*

Once a working copy of the ***DERIVE*** diskette is made as described in Section 1.3, ***DERIVE*** is easily started as follows:

- Turn on and boot your computer in the ordinary way from either a hard disk or a DOS system diskette in drive A. If you do not know how to boot your computer, study the manuals that came with the computer and/or get help from someone experienced in using the computer.

- If your working copy of ***DERIVE*** is on a diskette, insert the diskette in drive A, and at the DOS prompt enter the command

  ```
  A:DERIVE
  ```

  If ***DERIVE*** is on a DOS system diskette, you can boot your computer and start ***DERIVE*** from the same diskette.

- If your copy of ***DERIVE*** is on a hard disk, at the DOS prompt enter the command

  ```
  DERIVE
  ```

  If this does not start ***DERIVE***, you will have to change to the drive and directory containing the file DERIVE.EXE before executing the command. For example, at the DOS prompt enter

  ```
  C:
  ```

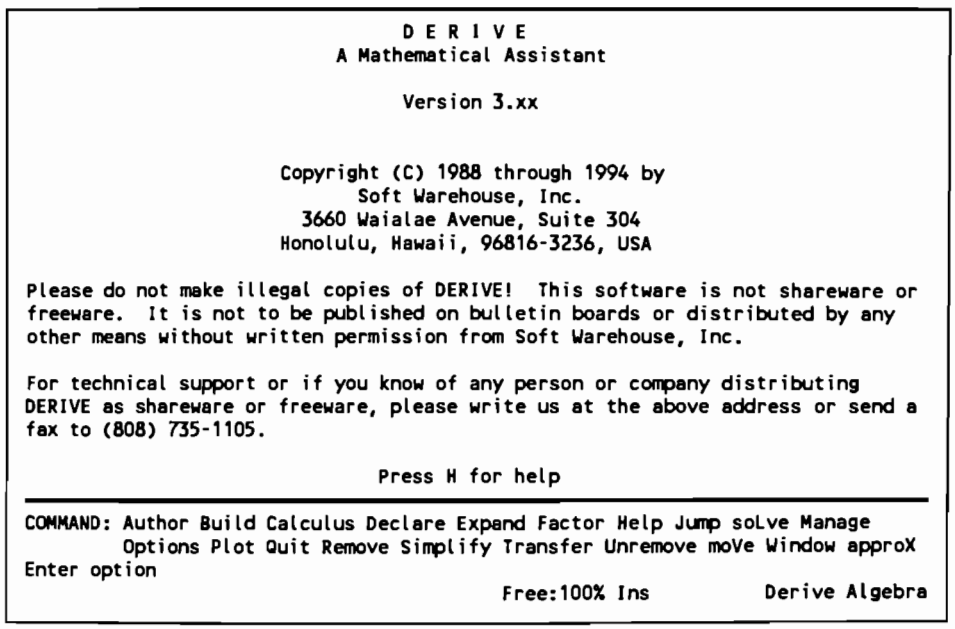
  and then

  ```
  CD C:\DERIVE
  ```

  to change to the C:\DERIVE subdirectory if one exists. See your computer's DOS manual for details on how to change the drive and directory. Once you have changed to the correct drive and directory, you must reissue the DERIVE command to actually start ***DERIVE***. If you are unable to start ***DERIVE*** from the hard disk, you can always start it from a diskette as described above.

Whichever method you use to start ***DERIVE***, there is a short delay while the program loads from the diskette or the hard disk. When the loading is complete, the initial ***DERIVE*** screen described in the next section is displayed.

2.2 The *DERIVE* Screen


D E R I V E
A Mathematical Assistant
Version 3.xx
Copyright (C) 1988 through 1994 by
Soft Warehouse, Inc.
3660 Waialae Avenue, Suite 304
Honolulu, Hawaii, 96816-3236, USA
Please do not make illegal copies of DERIVE! This software is not shareware or freeware. It is not to be published on bulletin boards or distributed by any other means without written permission from Soft Warehouse, Inc.
For technical support or if you know of any person or company distributing DERIVE as shareware or freeware, please write us at the above address or send a fax to (808) 735-1105.
Press H for help
COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Enter option
Free:100% Ins
Derive Algebra

The Initial ***DERIVE*** Screen

The bottom line of the ***DERIVE*** screen is called the **status line**. It displays the current status of ***DERIVE*** as it varies during use. Initially it shows

- that 100% of the computer memory space initially available to store mathematical expressions is still available;
- that the line editor is in insert mode rather than overwrite mode (see Section 2.4.);
- that the product name is ***DERIVE***; and
- that the current window is an **algebra window** (see Section 2.13).

Note that the percentage of space available is *not* updated each time an expression is added or removed. Rather, it is updated only when ***DERIVE*** finds it necessary to recycle memory. While recycling memory, other processing is briefly suspended. If recycling occurs while simplifying an expression, a character-sized rectangular block

is displayed in reverse video at the left end of the message line. If recycling occurs while an expression is being displayed on the screen, the update of the display pauses briefly.

Immediately above the status line is the **message line**. It displays what ***DERIVE*** is doing or what you are expected to do next. Initially, this line displays the message Enter option, meaning that you are expected to issue a command from the menu of options.

The two lines above the message line are the top-level **command menu**. The word COMMAND is the command **menu title**. The words following it on the same and following line are the command **menu options**.

2.3 Using The Menu System

One of the menu options is highlighted in reverse video. Each time you press the **Space** bar at the bottom of the keyboard or the **Tab** key, the highlight moves to the next option. Each time you press **Backspace** or **Shift-Tab**, the highlight moves to the previous option. (Section 1.6 describes how to type shift keys such as **Shift-Tab**.) To issue a command, first highlight the desired menu option and then press the **Enter** key.

If you are familiar with the keyboard, a faster way to issue a command is to type the capitalized letter in the menu option. This is the first letter of the option unless there is another option having the same first letter. It does not matter whether you type the letter in upper or lower case.

Many commands invoke sub-command menus, which may in turn invoke sub-sub-command menus, and so on. For brevity, this manual refers to a sequence of commands entered from successive menus by concatenating the menu option names. For example, this manual uses the phrase "issue the **Window Open** command" instead of the phrase "issue the **Window** command and then issue the **Open** sub-command".

After looking at a sub-command menu, you can return to the next higher-level command menu by pressing the **Esc** key. **Esc** is also used to abort computations that are taking too long.

Some menu option commands display one or more **selection fields**, each with its own name, followed by a colon and a list of two or more selections. For example, if you issue the **Options Display** command, it displays

```
Mode: Text Graphics  Reso: Medium (High)  Text:(Large) Small  Set: Std (Extended)
Adapter: MDA (CGA) EGA MCGA VGA Hercules AT&T T3100 PCjr Other
```

These lines consist of five selection fields: Mode, Reso(lution), Text, Set, and Adapter. Initially, the current display mode (either Text or Graphics) is highlighted. The highlight indicates that the Mode field is the **active field**. The current selections in the other fields are parenthesized. The parentheses indicate that these fields are inactive.

Press the **Tab** key to move the active field right. Press **Shift-Tab** to move the active field left. To make a selection, press **Space** or **Backspace** to highlight the desired selection or type the first capitalized letter in the selection's name. The latter method of making a selection also moves the active field right. Press **Enter** to invoke your selections. Alternatively press **Esc** to abort the command without making any changes.

Some menu option commands display one or more **data entry fields**, each with its own name, followed by a colon. Some commands display both selection *and* data entry fields. For example, if you issue the **Options Notation** command, it displays

OPTIONS NOTATION: Style: Decimal Mixed **Rational** Scientific Digits: 6

This line consists of a selection field (Style) and a data entry field (Digits). Initially, the current Style field selection (Rational) is highlighted indicating that the Style field is the active field. Press **Tab** to make the data entry field the active field, as indicated by a blinking cursor in the field. When a data entry field is active, you can edit or overwrite the value displayed in the field.

If you press a key for which there is no command, ***DERIVE*** produces a short **error beep**. If you find the beep annoying, use the **Options Mute** command to turn it off.

Now is a good time to start ***DERIVE*** if you have not already done so. You will probably find it easiest to learn by interactive participation while reading this manual. Alternatively, you may want to skim through the manual once to get an overview of ***DERIVE*** before starting to use the system.

2.4 Line Editing

Often while using ***DERIVE*** you will need to enter and/or edit a line of text. For example, try issuing an **Author** command by pressing **Enter** when the Author menu option is highlighted. A blinking cursor should appear and the system will wait for you to type a line of text.

Try typing your whole name on the line, but do *not* press **Enter**. If you make a mistake, press **Backspace** or **Ctrl-H** to delete the mistake and then type the correction. (Section 1.6 describes how to type control keys such as **Ctrl-H**.) Try deleting the last few characters of your name and then retyping them.

To make a change near the beginning of the line without deleting the characters at the end, press the **Left arrow key** or **Ctrl-S** to move the cursor to the left without erasing characters. If you move the cursor too far to the left, press the **Right arrow key** or **Ctrl-D** to move the cursor to the right.

A **token** is a string of letters or numeric characters terminated by a space, by the end of the line, or by some other type of character. Press **Ctrl-Left arrow** or **Ctrl-A** to move the cursor a token to the left. Press **Ctrl-Right arrow** or **Ctrl-F** to move the cursor a token to the right. Press **Home**, **Ctrl-]**, or **Ctrl-Q S** (i.e. press **Ctrl-Q** and then press **S**) to move to the left end (beginning) of the line. Press **End**, **Ctrl-**, or **Ctrl-Q D** to move to the right end of the line.

Press **Ctrl-U** to insert the most recently entered line of text at the cursor.

If the cursor is within a sequence of characters, press **Del** or **Ctrl-G** to delete the character at the cursor. Press **Ctrl-T** to delete the token at and to the right of the cursor. If you have so muddled the line you are editing that you would like a fresh start, press **Ctrl-Y** to delete the whole line. Press **Ctrl-Q Y** to delete the right end of the line beginning with the character at the cursor. Press **Ctrl-Q Backspace** or **Ctrl-Q H** to delete the left end of the line.

The effect of typing characters when the cursor is positioned within an author line depends on the current author entry mode. In **insert mode**, characters at and to the right of the cursor are moved right to make room for a new character. In **overwrite mode**, the character at the cursor is replaced by a new character.

In insert mode, the abbreviation Ins appears on the status line. This word is absent from the status line in overwrite mode. On PC compatible computers the cursor is also slightly taller in insert mode than in overwrite mode.

You can toggle between the two author entry modes by pressing **Ins** or **Ctrl-V**. Try toggling between modes and typing digits in both modes.

Control characters are normally interpreted by ***DERIVE*** as line editing commands, as described above. However, control characters can be entered on a line of text by pressing **Ctrl-P** before pressing the desired control character. For example, if you want to use the smiley face character (☺) as a variable name, press **Ctrl-P** then **Ctrl-A**.

Press **Esc** to escape from the **Author** command and return to the top-level command menu.

The line editing commands are summarized in an Appendix to this manual and in ***DERIVE***'s on-line help system (see Section 2.5).

2.5 Getting Help

Use the **Help** command to get help on using ***DERIVE***. To issue this command, press the **Space** bar until the Help menu option is highlighted and press **Enter**; or just type the letter **H**. While entering an expression on the author line (see Section 3.1), you can get help by pressing the **F1** function key.

The **Help** command displays the following menu:

```
DERIVE Help Menu

E - line Editing commands
F - Functions and constants
A - Algebra window commands
2 - 2D-plot window commands
3 - 3D-plot window commands
U - Utility file functions
S - current State of system
R - Return to DERIVE

Press letter for desired subject
```

The ***DERIVE*** Help Menu

Press the appropriate letter to get help on a particular subject as follows:

- Press **E** for help on the control key, function key, and alt key commands used for line editing (see Section 2.4).
- Press **F** for help on the pre-defined functions, operators, and constants that ***DERIVE*** knows how to simplify (see Chapter 6).
- Press **A** for help on the algebra window's menu option commands (see Chapters 3 and 4).
- Press **2** for help on the 2D-plot window's menu option commands (see Section 5.1).
- Press **3** for help on the 3D-plot window's menu option commands (see Section 5.2).
- Press **U** for help on the functions defined in the utility files included with ***DERIVE*** (see Chapter 9).

- Press **S** for a summary of the current settings of the system control variables (see Section 2.14) and the bytes of memory being used by ***DERIVE***.
- Press **R** to return to the top-level command menu.

The help includes a brief summary of each command or function and the section number in this manual where the command or function is described. Since the help for each subject requires several screens of text, the sub-command menu

HELP: **Next** Previous Resume

lets you scroll forward and backward through the text. Use the **Next** sub-command or press **PgDn** or **Ctrl-C** to see the next screen. Use the **Previous** sub-command or press **PgUp** or **Ctrl-R** to see the previous screen. Use the **Resume** sub-command to return to the top-level **Help** menu. Press **R** a second time to return to the ***DERIVE*** command menu.

The information displayed by the **Help** command is contained in the file DERIVE.HLP. ***DERIVE*** will be able to find this file if it is in the current directory, in the \DERIVE directory on the current drive, or in the directory from which ***DERIVE*** was invoked. If the **Help** command is unable to find the file DERIVE.HLP, it asks you to enter the drive and directory that contains the file. Press **Esc** if you do not know.

2.6 Quitting *DERIVE*

Use the **Quit** command to terminate ***DERIVE*** and return control to DOS. To issue this command, press the **Space** bar to highlight the Quit menu option and press **Enter**, or just type the letter Q. If you have entered any new expressions, before quitting, ***DERIVE*** waits for you to respond to the message

Abandon expressions (Y/N)?

If you type the letter Y for yes, ***DERIVE*** is terminated and the DOS prompt is displayed. Otherwise, the program continues as before.

2.7 Executing DOS Commands

Use the **Options Execute** command to execute a DOS command while running ***DERIVE***. It displays the current DOS drive and directory followed by the > prompt character. If you type a DOS command and press **Enter**, the command is executed just as if it were entered following the DOS prompt. For example, the command

```
DIR
```

displays the files in the current DOS directory. After the command is executed, press any key to return to the top-level command menu.

If you want to execute several DOS commands, issue the **Options Execute** command, then press **Enter** without typing any text. ***DERIVE*** is temporarily suspended so that you can execute any number of DOS commands. When you want to return to ***DERIVE***, enter the command

```
EXIT
```

at the DOS prompt.

When ***DERIVE*** is suspended it still occupies several hundred kilobytes of computer memory while the DOS commands are being executed. Thus commands that require a large amount of memory (such as DERIVE) should *not* be executed during the suspension. Before executing such commands, use the **Quit** command to terminate ***DERIVE***.

> **Warning**: Do *not* use the **Options Execute** command to load memory resident programs such as HERCULES.COM and GRAPHICS.COM. Doing so will abruptly abort ***DERIVE***. Memory resident programs must be loaded *before* starting ***DERIVE***.

Before the **Options Execute** command can execute a DOS command, ***DERIVE*** must locate the DOS command processor file COMMAND.COM. If it cannot locate the file, ***DERIVE*** displays the warning message

```
Cannot find COMMAND.COM
```

If running ***DERIVE*** off a diskette, this problem can be avoided by making the working copy of ***DERIVE*** a DOS system diskette (see Section 1.3.2).

2.8 Saving Expressions

You can save the mathematical expressions created during a ***DERIVE*** session as a file that can be examined and/or modified by any editor capable of editing standard DOS text files. The **Transfer Save** commands are used to save all *or* some of the expressions in the current algebra window as a file in various formats as follows:

- Use the **Transfer Save Derive** command to save expressions in a MTH file (math file) so they can be reloaded at a later time (see Section 2.9). Long expressions are written on multiple lines using a tilde (~) as the continuation character.
- Use the **Transfer Save Basic** command to save expressions in a BAS file (Basic source file). Expressions in BAS files are suitable for including in Basic programs.
- Use the **Transfer Save C** command to save expressions in a C file (C language source file). Expressions in C files are suitable for including in C programs.
- Use the **Transfer Save Fortran** command to save expressions in a FOR file (Fortran source file). Expressions in FOR files are suitable for including in Fortran programs. Expressions are restricted to columns 7 through 72, with long expressions written on multiple lines using a dollar sign ($) in column 6 as the continuation character.
- Use the **Transfer Save Pascal** command to save expressions in a PAS file (Pascal source file). Expressions in PAS files are suitable for including in Pascal programs. The file SUPPORT.PAS included on the ***DERIVE*** diskette defines several utility functions that may be called by the expressions saved in PAS files.

These commands prompt you to enter the drive, directory, name, and extension of the file to be saved. If you do not specify a drive and/or directory, the file is saved on the current DOS drive and directory. If you do not specify an extension, the file name extension MTH, BAS, C, FOR, or PAS is used as appropriate.

Use the **Transfer Save Options** command to choose the range (All or Some) of the expressions in the current algebra window to be saved by these commands. If the Some option is selected, you are prompted to specify the block of expressions you want saved each time one of the **Transfer Save** commands is issued.

Also use the **Transfer Save Options** command to examine and/or change the maximum line length used for MTH files. This line length does *not* affect BAS, C, FOR, or PAS files. Initially the line length is 79. The notation of numbers (rational vs. decimal) saved in MTH files is controlled by the **Options Notation** command (see Section 3.9).

2.9 Loading Expressions

You can load expressions read from MTH, DAT, and DMO files into the current algebra window using the following **Transfer** commands:

- Use the **Transfer Load Derive** command to *replace* the expressions in the current algebra window with those read from a MTH file.
- Use the **Transfer Merge** command to *append* the expressions read from a MTH file to those already in the current algebra window.
- Use the **Transfer Load Utility** command to more quickly load the function definitions and variable assignments in a MTH utility file (see Chapter 9). Although available for use, the definitions and assignments in the file are not displayed, thereby saving space.
- Use the **Transfer Load daTa** command to load a numeric DAT (data) file produced by instrumentation or other math programs. Each line of the data file must be a list of numbers separated by spaces and/or commas. The numbers must be in standard integer, rational, or floating point notation with an optional E or D followed by an exponent (e.g. -2.325E-7). ***DERIVE*** interprets each row of the data file as a row of a matrix, with blank lines used to separate matrices.
- Use the **Transfer Demo** command to *simplify* the expressions read from a DMO (demonstration) file. The demonstration files are listed in Section 1.1. After each expression is read and displayed in the current algebra window, the expression is simplified (see Chapters 3 and 4) and that result is also displayed. ***DERIVE*** then waits for you to press any key other than **Esc** before reading the next expression. If you press **Esc**, the demonstration is suspended and you are free to perform any ***DERIVE*** operation. To resume the demonstration from the point at which it was suspended, simply issue another **Transfer Demo** command.

These **Transfer** commands prompt you to enter the drive, directory, name, and extension of the file to be read. If you do not specify a drive and/or directory, the file is read from the current DOS drive and directory, in the \DERIVE directory on the current drive, or in the directory from which ***DERIVE*** was invoked. If you do not specify an extension, the file name extension MTH, DAT, or DMO is used, as appropriate.

As an alternative to typing in a file name, you can press the **F1** function key to display a list of files with the appropriate type name. Then use the arrow keys (**Up**, **Down**, **Left**, **Right**, **PgUp**, **PgDn**, **Home**, and **End**) or control keys (see Section 3.3) to highlight the desired file name and press **Enter**. You can also move the highlight by typing the first letter of the desired file name.

To display DOS directories other than the current directory, type the path name of the directory *before* pressing **F1**. To display files with a type name other than the default, type a period and the desired type name *before* pressing **F1**. Use an asterisk to indicate that you want files with any type name.

When using the **Transfer Load Derive** and **Transfer Merge** commands, if you press **Ctrl-Enter** instead of **Enter** after typing the file name, each expression is simplified (see Section 3.2) as it is read in. If you save a series of expressions in a MTH file, this feature allows you to simplify the expressions in batch mode, say overnight.

You can also load one or more MTH, DAT, or DMO files when starting ***DERIVE*** by including the file names following the word DERIVE on the DOS command line. For example, to start ***DERIVE*** and automatically run the demonstration file ARITH.DMO, enter the command

```
DERIVE ARITH
```

at the DOS prompt.

By default, a file with an extension name of MTH is loaded as a math file, DAT as a data file, and DMO as a demonstration file. To force a file to load in a particular way, follow the file's name with /M to load it as a math file, with /U to load it as a utility file, with /T to load it as a data file, or with /D to load it as a demonstration file. For example, the command line

```
DERIVE NUMBER/U PLOT2D/M ALGEBRA/D
```

loads ***DERIVE***, loads NUMBER.MTH as a utility file, loads PLOT2D.MTH as a math file, and then loads ALGEBRA.DMO as a demonstration file.

Use the **Transfer Clear Expressions** command to clear all expressions from the current algebra window (see Section 3.1).

Use the **Transfer Clear Variables** command to clear all user-assigned variable values and domains (see Sections 4.10 and 4.11).

Use the **Transfer Clear Functions** command to clear all user-defined functions (see Section 4.12).

Use the **Transfer Clear All** command to clear all expressions from the current algebra window, all user-assigned variable values and domains, and all user-defined functions.

2.10 Printing Expressions

Use the **Transfer Print Expressions** commands in this section to send some *or* all of the expressions in the current algebra window to a printer or to save them as an ASCII text file. The expressions are printed or saved in two-dimensional format using raised exponents and built-up fractions. Expressions too long to be printed on a single line are broken and printed on multiple lines. The annotations associated with the expressions are also printed or saved.

- Use the **Transfer Print Expressions Printer** command to send the expressions in the current algebra window to the system printer. If the printer is not on-line and ready to receive input, the warning message Printer not ready is displayed.

- Use the **Transfer Print Expressions File** command to send the expressions in the current algebra window to a file. This command prompts you to enter the drive, directory, name, and extension of the file to be written. If you do not specify a drive and/or directory, the file is saved on the current DOS drive and directory. If you do not specify an extension, the file name extension PRT is used to indicate that the file is a file formatted for printing. Unlike MTH files, PRT (print) files are *not* capable of being read by **Transfer Load** and **Transfer Merge** commands.

- Use the **Transfer Print Expressions Options** command to set various options that affect the printing of expressions. It displays the initial values and prompts you to enter new values for the range (All or Some) of expressions to print, the decision to print annotations (Yes or No), and the ASCII character set (Standard or Extended) used to print expressions. Initially all expressions are printed, annotations are printed, and the extended character set is used if the Laserjet, Dotmatrix, or deskjetC (color) printer type is selected (see below).

The standard ASCII character set consists of 128 characters. The extended character set consists of the standard set plus an additional 128 characters containing some of the common Greek letters and mathematical symbols. If your printer is capable of printing them, selecting the extended character set produces more legible and attractive printouts.

Use the **Transfer Print Options** and **Transfer Print Layout** commands to set various options that affect both the printing of expressions *and* screen images (see Section 2.11).

The **Transfer Print Options** command displays the initial values and prompts you to enter new values for the following items:

- Printer type - Plain, Laserjet (Model I, II, III, or IV), Dotmatrix, or deskjetC (Model 500C or 550C);

- Font print size - Small or Large;

- Print orientation - Landscape or Portrait;
- Paper size - Standard or Legal; and
- Background color - White or Black.

The **Transfer Print Layout** command displays the initial values and prompts you to enter new values for the length, width, and margins of the print page. Initially, the page length is 66 lines and the width is 80 characters; the top and bottom margins are 0 lines; and the left margin is 8 characters and the right is 3.

By default, ***DERIVE*** sends printer output to port LPT1. If printers are assigned to ports LPT2 or LPT3, ***DERIVE*** printer output can sent to these ports by changing the printer port assignment in the ***DERIVE*** initialization file (see Section 2.14).

2.11 Printing Screen Images

Use the **Transfer Print Screen** commands in this section to send a graphics screen image of all *or* part of the ***DERIVE*** screen to a printer or to save it as a graphics image file. The options set by the **Transfer Print Options** and **Transfer Print Layout** commands (see Section 2.10) also affect the printing and saving of screen images.

Graphics screen images are printed or saved in the same resolution as that of the current display mode. Thus, use the **Options Display** command (see Section 2.12) to select a high resolution graphics display mode before printing or saving screen images.

- Use the **Transfer Print Screen Printer** command to send a graphics image of all or part of the ***DERIVE*** screen to the system printer. Make sure the printer is on and ready to receive input before issuing this command.

  Press **Esc** to abort a print-out that is in progress. But be patient: depending on the printer and display mode (see Section 2.12), it may take several seconds to a minute or two to print a graphics screen image. ***DERIVE*** beeps when a print-out is complete.

  To print a graphics screen image, the printer type selected using the **Transfer Print Options** command (see Section 2.10) should be either Laserjet, Dotmatrix, or deskjetC. If Laserjet or Dotmatrix is selected, black and white images are sent to the printer so the same background color must be used in all regions being printed (see Section 2.12).

  If deskjetC (i.e. a DeskJet 500C, 550C, or compatible color printer) is selected, eight-color images are sent to the printer so different background colors can be used in the regions being printed. However, a white background (i.e. color number 15) is recommend when using a Deskjet color printer since much less ink is required. To make a black and white screen image using any Deskjet printer, select the Laserjet printer type.

  If the screen image is too large to fit entirely on a page of paper, try selecting Landscape orientation and Small print size using the **Transfer Print Options** command (see Section 2.10).

Graphics screen images can be sent to the printer at any time by pressing **Shift-F10** or **Shift-F9**. **Shift-F10** sends an image of the entire ***DERIVE*** screen. **Shift-F9** sends an image of just the current window (algebra, 2D-plot, or 3D-plot window). Using these keys instead of the **Transfer Print Screen Printer** command has the advantage of not disturbing the screen image with the menus and prompts that are displayed by the command.

Uncompressed TIF (Tagged Image Format) files are a popular format for storing graphics images. Many popular word processing and desktop publishing programs can incorporate TIF file images into documents.

- Use the **Transfer Print Screen File** command to save a graphics image of all or part of the ***DERIVE*** screen as a file in uncompressed, black and white TIF file format.

  Depending on the display mode (see Section 2.12), it may take several seconds to a minute or two to save a graphics screen image as a TIF file. ***DERIVE*** beeps when the screen image has been saved.

  Make sure that the *same* background color is used for all regions being saved (see Section 2.12). Note that the printer type does *not* matter when saving screen images as TIF files.

Graphics screen images can be saved as a TIF file at any time by pressing **Ctrl-F10** or **Ctrl-F9**. **Ctrl-F10** saves an image of the entire ***DERIVE*** screen. **Ctrl-F9** saves an image of just the current window (algebra, 2D-plot, or 3D-plot window). Using these keys instead of the **Transfer Print Screen File** command has the advantage of not disturbing the screen image with the menus and prompts that are displayed by the command.

- Use the **Transfer Print Screen Options** command to set various options that affect the printing or saving of screen images. It displays the initial values and prompts you to enter new values for the region (All, Window, or Current) to print or save, the position on the page to print the image (Center or Layout), and the name to use for the next TIF file. Integer suffixes are attached to the name as necessary to make the name of each TIF file unique. If Layout is selected, the image is positioned on the page using the left and top margins specified by the **Transfer Print Layout** command.

An alternative way to print screen images is by pressing **Shift-PrtSc** on PC compatible computers or by pressing the **Copy** key on PC-9801 compatible computers. In text display mode (see Section 2.12), this method can be used to print a screen image on virtually any printer. However, to print a screen image in graphics display mode, a screen capture program appropriate for the display mode and your printer must be installed as a resident program *before* starting ***DERIVE***.

An appendix to this manual lists several screen capture programs that support a wide variety of printers and display modes. Also listed are some popular word processing programs that include utility programs for incorporating screen images into documents.

2.12 Display Modes and Colors

This section summarizes the resolution and number of colors provided by the various display modes and how to change the display mode.

Use the **Options Display** command to switch between the various display modes. On PC compatible computers, it displays five selection fields similar to

```
Mode: Text Graphics  Reso: Medium (High)  Text:(Large) Small  Set: Std (Extended)
Adapter: MDA (CGA) EGA MCGA VGA Hercules AT&T T3100 95LX PCjr Other
```

The highlight in the Mode field and parentheses in the other fields indicate the current display mode.

Use the Mode field to select either Text or Graphics display mode. In either mode, both text and plots can be displayed. However, in Text mode each pixel (plot point) is the size of a character. Since Text mode gives such low resolution, plots are better done in Graphics mode. However, in Text mode screen updates are faster and characters may be sharper and more legible.

Use the Reso(lution) field to select either Medium or High resolution display mode. This field has no effect when using MDA or Hercules compatible display adapters. In High resolution, 80 characters or 640 pixels can be displayed on each screen line. In Medium resolution, only 40 characters or 320 pixels can be displayed and some command menus must be abbreviated. The primary advantages of Medium resolution are that: a) characters are larger and easier to read, b) screen updates are faster, and c) when using CGA display adapters more colors are available to distinguish plots.

Use the Text field to select either Large or Small text mode. In Small text mode characters are half-height and therefore enable you to view more expressions simultaneously. Small text mode is available only on computers with EGA or VGA graphics capability. The Text field selection has no effect for computers without such graphics capability.

Use the Set field to select either the Std (standard) or Extended ASCII character set for displaying window borders and expressions on the screen. If window borders and/or expressions do not display correctly on the screen, try selecting the standard character set.

Use the Adapter field to select which type of adapter to use. A display adapter is the circuitry in the computer that produces the video image sent to the display monitor. Some computers provide more than one type of adapter, so you may want to try various adapters to see which ones work on your computer. ***DERIVE*** supports the following display adapters on PC compatible computers:

- **MDA** is for computers with a Monochrome Display Adapter. The three "colors" this adapter provides are black, dim white, and bright white.
- **CGA** is for computers with a Color Graphics Adapter.
- **EGA** is for computers with an Enhanced Graphics Adapter. This adapter can also display CGA graphics modes. If you select graphics mode and this adapter, the **Options Display** command displays a second selection field similar to

  Monitor: **Color** Enhanced Monochrome

  If you have a monitor that can display 350 pixel lines, select the Enhanced monitor; otherwise, select the Color or Monochrome monitor as appropriate.
- **MCGA** is for PS/2 models 25 and 30 computers with a Multi-Color Graphics Array adapter. This adapter can also display CGA graphics modes.
- **VGA** is for computers with a Video Graphics Array adapter. This adapter can also display CGA, EGA, and MCGA graphics modes.
- **Hercules** is for computers with a Hercules compatible graphics adapter. To select this adapter execute the DOS command HERCULES *before* starting ***DERIVE***. HERCULES.COM is a memory resident program on the ***DERIVE*** diskette. If a resident screen capture program is also to be installed (see Section 2.10), you may need to load it *after* loading HERCULES.COM and *before* starting ***DERIVE***.
- **AT&T** is for computers with an AT&T 6300 compatible graphics adapter. This adapter can also display CGA graphics modes. If when using this adapter in graphics mode, the window borders and/or expressions do not display correctly on the screen, use the Set field of the **Options Display** command to change to the standard ASCII character set.
- **T3100** is for Toshiba 3100 and 5100 computers with a Toshiba 3100 compatible graphics adapter. This adapter can also display CGA and EGA graphics modes.
- **95LX** is for the 240 by 128 pixel screen Hewlett Packard 95LX, 100LX, or 200LX Palmtop computer.
- **PCJr** is for IBM PC-Junior computers with a PC-Junior graphics adapter. This adapter can also display CGA graphics modes.
- **Other** is for non-standard video modes such as those provided by some Super VGA graphics adapter cards. It displays the current video mode number and prompts you to enter a new mode number between 0 and 255 inclusive. In order for ***DERIVE*** to use a non-standard video mode, the mode must be fully supported by BIOS software included with your graphics card. The card's manual should list the video modes it supports and explain how to load the BIOS software.

The following table summarizes the resolution and number of colors provided by the various display adapters:

Adapter	Resolution	Columns	Rows	Colors
MDA	--	80	25	3
CGA	Medium	320	200	4
CGA	High	640	200	2
EGA	Medium	320	200	16
EGA	High	640	350	16
MCGA	--	640	480	2
VGA	--	640	480	16
Hercules	--	720	348	2
AT&T	--	640	400	2
T3100	--	640	400	2
95LX	--	240	128	2
PCJr	Medium	320	200	16
PCJr	High	640	200	4

Display Adapter Table

The result of the **Options Display** command is unpredictable if you select an adapter or monitor that is incompatible with your computer's actual hardware. After setting the display mode, you can reissue the **Options Display** command to see if the resulting display mode is what you requested.

On PC compatible computers, pressing the **F5** function key switches to the previous display mode. If you frequently switch between algebra and plot windows, **F5** makes it easy to switch between text and graphics display modes. Note that both the current and previous display modes are saved in the initialization file created by the **Transfer Save State** command (see Section 2.14).

Exercise: If running ***DERIVE*** on a PC compatible computer having a graphics display adapter, use the **Options Display** command to switch to the highest resolution possible on your computer. Then use the **Options Display** command to switch back to text mode. Finally use the **Transfer Save State** command to save these display modes in the DERIVE.INI initialization file so that **F5** switches between text and graphics modes whenever you use ***DERIVE***.

Use the **Options Color Work** command to change the color used to display mathematical expressions in an algebra window. This command also controls the background color used for algebra, 2D-plot, and 3D-plot windows.

Use the **Options Color Menu** command to change the color used to display command menu options, message line text, and the status line text (see Section 2.2). This command also controls the background color used for the menu and border areas of the ***DERIVE*** screen.

The **Options Color** commands display the integers 0 through 15 on the message line, each number displayed in its respective color, and prompt you to enter a color number in each of their respective selection fields. You can enter any integer from 0 to 15, however the actual number of colors available depends on your computer's graphics hardware and the current display mode.

On monochrome monitors different colors are often represented as different brightnesses. You may have to adjust the brightness and contrast knobs in order to distinguish the various shades. Some colors may map to the same brightness as the background, making text or plots invisible. If this should occur, make the background color 0 and the foreground colors 15.

Exercise: If running ***DERIVE*** on a PC compatible computer having a color monitor, use the **Options Color** commands to change to your favorite foreground and background colors for the work and menu areas. Then use the **Transfer Save State** command to save these color selections in the DERIVE.INI initialization file.

2.13 Window Commands

DERIVE's windowing commands make it possible to open multiple algebra and plot windows. Windows can be overlaid and/or split.

Multiple algebra windows allow you to pursue more than one derivation stream since their histories fork when you split an algebra window. Multiple algebra windows also allow you to freeze the display of particularly important expressions in one window while scrolling expressions in another.

Multiple plot windows allow you to maintain separate lists of expressions to plot. They also allow you to simultaneously plot expressions using different scales and coordinates.

Use the **Window Split Horizontal** or **Window Split Vertical** command to split a window. These commands prompt you to enter the number of the row or column at which to make the split. This number is measured relative to the upper left corner of the active window (i.e. row 1 and column 1). The default row or column number splits the window in half.

When deciding whether to make a horizontal or vertical split and what row or column to split, consider the orientation and relative window size that will provide the best display for both your expressions and plots.

Each window on a split screen has a unique **window number** displayed in the upper left corner. The number of the currently **active window** is highlighted using reverse video.

Use the **Window Designate** command to change the type of the active window. This command displays the selection field

WINDOW DESIGNATE: Type: **Algebra** 2D-plot 3D-plot

and waits for you to select a window type. The type of the current window will be highlighted.

Use the **Window Open** command to overlay a new window on the active window. This command also prompts you to select a window type for the new window. Note that overlaid windows share the same window number and that only the top window can be active.

Use the **Window Close** command to close a window. This command prompts you to enter the window number of the window to close. The default is the active window. Closing a window frees the valuable memory and screen space used by that window.

There are several **Window** commands for changing the active window and for flipping between overlaid windows:

- Use the **Window Goto** command to make a specific window the active window. This command prompts you to enter a window number. The default is the next window number.
- Use the **Window Next** command or press **F1** to make the next window active. Note that this takes less effort but is less flexible than the **Window Goto** command.
- Use the **Window Previous** command or press **Shift-F1** to make the previous window active.
- Use the **Window Flip** command or press **F2** to flip to the next overlaid window.
- Press **Shift-F2** to flip to the previous overlaid window.

2.14 Saving Current State

As described in this and subsequent chapters, there are many system control settings in ***DERIVE*** that you can change. They include the following:

- numeric precision, notation, and radix base (Sections 3.8, 3.9, and 3.11),
- logarithm, exponential, and trigonometric simplification modes (Chapter 6),
- complex branch selection (Section 4.5),
- ordering and default domain of variables (Section 4.3 and 4.10),
- initial display mode and default colors (Section 2.12),
- variable name and case-sensitive input modes (Section 4.1),
- display format and times operator output modes (Section 3.13),
- printer type, initialization string, and printer device name (Section 2.10),
- printer page layout and print options (Section 2.10),
- aspect ratio, accuracy, coordinate system, follow cross mode for 2D plots (Section 5.1), and
- number of grid lines, and top and bottom colors for 3D plots (Section 5.2).

Collectively called the **current state** of ***DERIVE***, these settings can be saved as an INI (initialization) file. An INI file can be examined and/or modified by any editor capable of editing standard DOS text files. Each line of an INI file consists of a state variable followed by its value.

Use the **Transfer Save State** command to save the current state as an INI file. This command prompts you to enter the drive, directory, name, and extension of the file to be written. If you do not specify a drive and/or directory, the file is saved on the current DOS drive and directory. If you do not specify an extension, the file name extension INI is used.

Use the **Transfer Load State** command to load the state saved in an INI file. This command prompts you to enter the drive, directory, and name for the file to be read. If you do not specify a drive and/or directory, the file is read from the current DOS drive and directory. If you do not specify an extension, the file name extension INI is used.

When ***DERIVE*** starts, it automatically loads the state initialization file DERIVE.INI if there is such a file in the current directory, in the \DERIVE directory on the current drive, or in the directory from which ***DERIVE*** was invoked.

Thus if you want a particular state each time you start ***DERIVE***, use the various state setting commands (e.g. **Options Precision**, **Options Display**, and **Manage Branch**) to set the desired initial state. Then use the **Transfer Save State** command to save the initialization file DERIVE.INI in the same directory as DERIVE.EXE so DERIVE.INI will be loaded regardless of which directory you are in when you start ***DERIVE***.

By default, ***DERIVE*** sends printer output to port LPT1. To redirect ***DERIVE*** output to ports LPT2 or LPT3, use a text editor to change the printer port assigned to *PRINTER-DEVICE* in DERIVE.INI. If the file DERIVE.INI does not already exist, you can create one using the **Transfer Save State** command.

You may want ***DERIVE*** to start in one of several different initial states. For example, if you have both a dotmatrix and a LaserJet printer, you can have initialization files set for each printer type. If DERIVE is immediately followed by the name of an INI file on the DOS command line or in a BAT file (DOS batch file), ***DERIVE*** is initialized using this file instead of DERIVE.INI. For example, the command line

```
DERIVE LASERJET.INI ARITH.DMO
```

initializes ***DERIVE*** using the settings in LASERJET.INI, and then runs ARITH.DMO.

Chapter 3. Arithmetic

This chapter describes how to enter, build, simplify, move, and remove numeric expressions. For a demonstration of some of these capabilities, load the file ARITH.DMO using a **Transfer Demo** command (see Section 2.9).

Most of the ***DERIVE*** menu option commands are inapplicable until there are some expressions on the screen. So let's begin by learning how to enter numeric expressions.

3.1 Entering Expressions

Use the **Author** command to enter a new expression. This command displays the prompt AUTHOR expression and waits for you to type an expression. Section 2.4 describes how to type and edit a line of text.

When satisfied with the line of text, press the **Enter** key to actually enter the expression. The entire line of text is entered regardless of where the cursor is when you press **Enter**.

Exercise: Start ***DERIVE*** (see Section 2.1), issue an **Author** command, and enter the decimal number

```
12.345
```

If the author line is syntactically correct, the screen should look similar to:

```
#1:  12.345

COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
         Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Enter option
User                                    Free:100%              Derive Algebra
```

Note that:

- The expression 12.345 is **highlighted** in reverse video;
- The expression's unique **label number** is displayed along the left edge of the window, followed by a colon; and
- The annotation, User, displayed at the left end of the status line, indicates that the highlighted expression was entered by you, the user.

While entering an expression on the author line, you can press the F1 function key to access the ***DERIVE*** help system (see Section 2.5). Help is available on line editing commands and ***DERIVE***'s functions and operators. Upon exiting the help system, your text on the author line is restored with the cursor at its original location.

If an author line contains a syntax error, ***DERIVE*** displays

Syntax error detected at cursor

emits an error beep, and positions the cursor where the error was detected. The error may actually have occurred anywhere to the *left* of this point on the line. You can edit the author line as described in Section 2.4 and then press **Enter** to try again.

Exercise: Issue an **Author** command, type the intentional syntax error

```
12.34.5
```

and press **Enter**. You should hear an error beep and the cursor should be positioned at the second decimal point. Correct the error by deleting the decimal point and reenter the expression.

Standard symbols are used for the four basic mathematical operators:

- + (plus) to denote addition;
- – (minus) to denote subtraction or negation;
- * (times) to denote multiplication; and
- / (divide) to denote division.

These binary infix operators are placed between their operands. + and – can also be unary prefix operators and be placed before a single operand.

You can use parentheses, ( ), to control the order in which operators are applied. Brackets, [ ], and braces, { }, are *not* equivalent to parentheses and can *not* be used to control the order of application of operators *or* to enclose the arguments of functions (see Section 3.4).

* and / have a higher operator precedence than + and –. In the absence of parentheses, operators with higher precedence are applied before operators with lower precedence. For example,

```
3 + 4 * 5
```

is equivalent to

```
3 + (4 * 5)
```

rather than

```
(3 + 4) * 5
```

+ and – have equal precedence, as do * and /. In the absence of parentheses, operators with equal precedence are applied left to right. For example,

```
1 / 2 * 3
```

is equivalent to

```
(1 / 2) * 3
```

rather than

```
1 / (2 * 3)
```

Instead of using * to indicate multiplication, you can separate two expressions with one or more spaces and/or parentheses to implicitly indicate multiplication. For example,

```
2 (3 + 5)
```

is equivalent to

```
2 * (3 + 5)
```

Sometimes books print rational numbers as an integer followed by the fractional part. For example, 3.5 is printed as 3½. This is short for 3 plus ½, *not* 3 times ½. In ***DERIVE***, use the + operator between the integer and fractional part of a number to prevent it from being interpreted as implicit multiplication. For example, enter 3½ – 2¼ as

```
(3+1/2) – (2+1/4)
```

Use the ^ operator to indicate exponentiation (raising an expression to a power). For example, 3 squared (i.e. 3^2) is entered as

```
3^2
```

^ has a higher precedence than +, –, *, and /. Thus operands of ^ that include these operators should be enclosed in parentheses. For example, the cube root of 9 (i.e. $9^{1/3}$) must be entered as

```
9^(1/3)
```

rather than

```
9^1/3
```

Unlike other operators, ^ associates to the right. Thus in the absence of parentheses, successive ^ operators are applied right to left. For example,

```
4^3^2
```

is equivalent to

```
4^(3^2)
```

rather than

```
(4^3)^2
```

The ^ operator can also be used to enter numbers in scientific notation. For example, the number 6.02 times 10^{23} can be entered as

```
6.02 * 10^23
```

Use the % operator to indicate percent or one hundredth of an expression. For example,

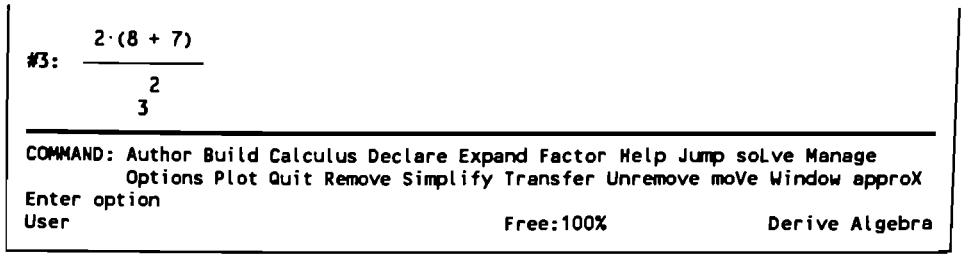
```
22% + 36%
```

is equal to 0.58. The % operator has a higher precedence than any operator discussed thus far.

Exercise: Issue an **Author** command and enter the expression

```
2 (8 + 7) / 3^2
```

The screen should look similar to:


#3: 2·(8 + 7) / 3^2
COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Enter option
User
Free:100%
Derive Algebra

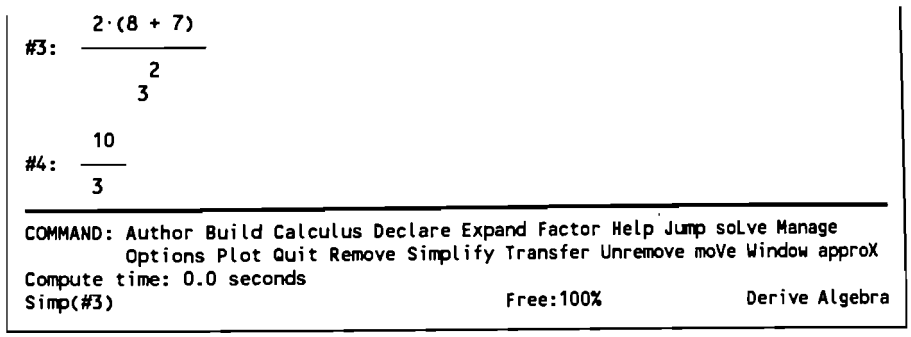
Note that:

- The expression is formatted using raised exponents and built-up fractions;
- The small centered dot between the 2 and the (8 + 7) in the numerator indicates multiplication; and
- The expression is *not* automatically simplified. The next section describes how to simplify expressions.

3.2 Simplifying Expressions

Use the **Simplify** command to simplify an expression. This command prompts you to enter the label number of the expression to be simplified. The default is the number of the expression highlighted in the window. After you press **Enter**, ***DERIVE*** simplifies the expression and displays the result.

Exercise: Simplify the expression in the last example. The screen should look similar to:


#3: 2·(8 + 7) / 2 / 3
#4: 10 / 3
COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Compute time: 0.0 seconds
Simp(#3)
Free:100%
Derive Algebra

Note that:

- The expression is simplified to 10/3 in almost no time (i.e. in less than 1/20th of a second);
- The simplified result is highlighted and has its own label number; and
- The annotation, Simp(#3), displayed at the left end of the status line indicates that the highlighted expression was derived by simplifying expression #3.

If you have no interest in seeing the unsimplified version of an expression, you can enter and simplify an expression in a single step. After entering an expression, if you press **Ctrl-Enter** or **Ctrl-J** instead of **Enter** to enter the expression, only the simplified result is displayed in the window.

Exercise: Enter and simplify the expression 2+3 in a single step.

If you want to see both the input *and* the simplified result displayed on a single line, enter the expression on the author line followed by an equal sign (=) and press the **Enter** key. ***DERIVE*** will then display an equation whose left side is the unsimplified form of the expression and whose right side is its simplified form. For example, if you enter `2^3=`, it will display as

$$2^3 = 8$$

3.3 Highlighting Expressions

Frequently you will want to use an existing expression or subexpression as part of a new expression. Instead of retyping it, you can select the expression by highlighting it. This section describes several ways to highlight expressions.

Use the **Jump** command to highlight a given expression. This command prompts you to enter the label number of the expression to be highlighted. Alternatively, you can highlight an expression or subexpression using the **direction keys** as follows:

- **Up arrow** or **Ctrl-E** moves the highlight up one expression. The expressions scroll if necessary to bring the highlighted expression onto the screen.
- **Down arrow** or **Ctrl-X** moves the highlight down one expression.
- **PgUp** or **Ctrl-R** moves the highlight up about one half screen.
- **PgDn** or **Ctrl-C** moves the highlight down about one half screen.
- **Ctrl-PgUp** or **Ctrl-Home** moves the highlight up to the first expression.
- **Ctrl-PgDn** or **Ctrl-End** moves the highlight down to the last expression.

Exercise: Use the direction keys to highlight the unsimplified expression #3 that you entered in Section 3.1.

You can also highlight subexpressions of an expression:

- **Right arrow** moves the highlight through the successive operands of an operator. For example, you can sequence through the terms of a sum or the factors of a product. If an expression is wider than the window, the expression scrolls horizontally as necessary to include highlighted portions. **Ctrl-D** is equivalent to **Right arrow** when you are *not* line editing.
- **Left arrow** moves the highlight back through the operands of an operator. **Ctrl-S** is equivalent to **Left arrow** when you are *not* line editing.
- Within a subexpression, **End** moves the highlight to the rightmost operand of a subexpression. For example, it moves the highlight to the *last* term of a sum.
- Within a subexpression, **Home** moves the highlight to the leftmost operand of a subexpression. For example, it moves the highlight to the *first* term of a sum.
- Within a subexpression, **Down arrow** or **Ctrl-X** moves the highlight down one level into the operands of an operator. For example, you can descend to the factors of a term in a sum.

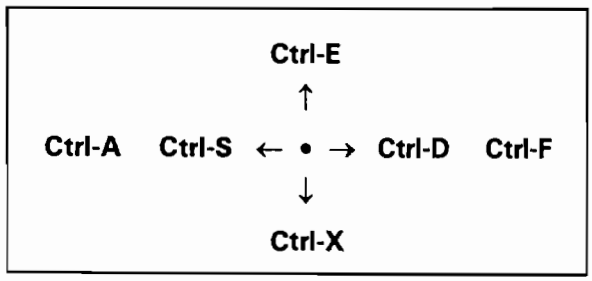
- Within a subexpression, **Up arrow** or **Ctrl-E** moves the highlight up one level back towards the entire expression.

Exercise: Use the direction keys to highlight the 8+7 subexpression in the unsimplified expression #3 that you entered in Section 3.1.

If the highlighted expression is wider than the algebra window, use **Ctrl-Right arrow** or **Ctrl-F** to scroll the expression horizontally to the left in order to see the rest of it. Use **Ctrl-Left arrow** or **Ctrl-A** to scroll the expression horizontally to the right. Unlike moving the highlight using **Right arrow** and/or **Left arrow**, horizontal scrolling does *not* move the highlight within the expression.

Exercise: On the author line, enter and simplify the expressions 99^1, 99^2, 99^3, etc. until you see a pattern in the last two digits of the results. Then enter and simplify 99^{99}. The result will be a number too big to display all the digits on a single line of the screen. Use control keys to scroll the number left on the screen to verify that the last two digits are what you expect.

You may prefer moving the highlight using control keys rather than direction keys because control keys can be entered without moving your hands from the keyboard. Note that the control keys form an easily remembered cross or diamond pattern on the keyboard that is suggestive of their roles:



The Control Key Diamond

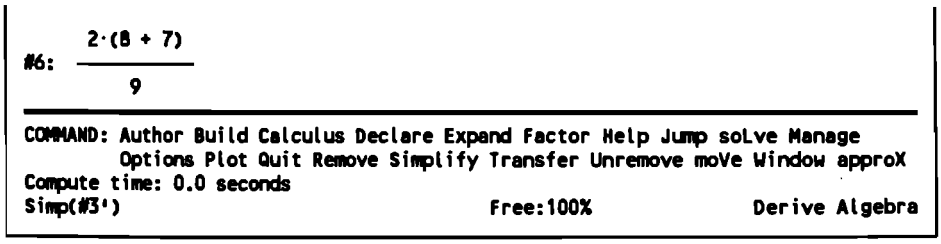
Note that the left and right arrow keys and the **Home** and **End** keys have a dual use. In subexpression mode, these keys move the highlight within an expression as described above. In line-edit mode, these keys move the cursor while editing a line of text (see Section 2.4). When entering an expression on the author line, press the **F6** function key to toggle between these two modes. In line-edit mode, Lin appears on the status line at the bottom of the ***DERIVE*** screen. In subexpression mode, nothing appears at this place on the status line.

When an **Author** command is issued, the initial arrow key mode (line-edit or subexpression) is controlled by a field of the **Options Input** command (see Section 4.1). The initial arrow key mode is saved as part of the *DERIVE* state in INI files (see Section 2.14).

Exercise: Issue an **Author** command and type your whole name, but do *not* press **Enter**. Press the **F6** key several times and note the appearance and disappearance of Lin on the status line. In line-edit mode, use the arrow, **Home**, and **End** keys to move the cursor on the author line. Then press **F6** to switch to subexpression mode and note the effect these keys now have on the highlight. Press **Esc** when you have mastered the dual use of these keys.

When a subexpression is highlighted, the **Simplify** command produces a copy of the whole expression in which only the highlighted portion is simplified.

Exercise: Simplify only the 3^2 subexpression in the unsimplified expression #3. The screen should look similar to:


#6: 2·(8 + 7) / 9
COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Compute time: 0.0 seconds
Simp(#3')
Free:100%
Derive Algebra

Note that:

- The label number of the result will vary depending on how many expressions you have entered thus far.
- The single quote mark in the annotation, Simp(#3'), indicates that only part of expression #3 was simplified.

While typing an expression on the author line, press the **F3** function key to insert whatever expression or subexpression is highlighted into the author line just before the cursor. Pressing the **F4** function key also inserts the highlighted expression into the author line but encloses it in parentheses. These function keys make it easy to edit previously entered expressions and subexpressions.

Exercise: Issue the **Author** command, highlight the result of the previous exercise if it is not already highlighted, press **F3** and change the denominator from 9 to 8, then enter the modified expression.

When typing an expression on the author line, previous expressions can be referenced by typing the expression's label number (see Section 3.1) preceded by the # character. Note that subexpressions of previous expressions can *not* be referenced in this way. However, the **F3** and **F4** function keys can be used to insert subexpressions on the author line.

Exercise: Enter and simplify the expression

```
#2 + #3
```

3.4 Entering Functions

A mathematical function is entered by typing its name followed by its arguments enclosed in parentheses and separated by commas. In case-insensitive mode (see Section 4.1), the function name can be entered using either upper-case or lower-case letters, but is displayed only in upper-case. The parentheses need not be entered if the function has only one argument *and* the argument is an integer, a decimal fraction, or a variable (see Section 4.1). Note that brackets, [ ], and braces, { }, are *not* equivalent to parentheses. Only parentheses can be used to enclose the arguments of functions.

DERIVE has many pre-defined mathematical functions (see Chapter 6). You can also define your own functions (see Section 4.12). The following are a few of the pre-defined functions especially relevant to arithmetic:

- GCD returns the greatest common divisor of its arguments. LCM returns the least common multiple of its arguments. For example,

  ```
  GCD(12, 30)
  ```

  and

  ```
  LCM(12, 30)
  ```

  simplify to 6 and 60, respectively.

- AVERAGE returns the arithmetic mean or average of its arguments. For example,

  ```
  AVERAGE(3, 7)
  ```

  simplifies to 5.

- ABS returns the absolute value of its argument. The absolute value function can also be entered and is displayed using vertical bars to delimit the argument. For example,

  ```
  |-8|
  ```

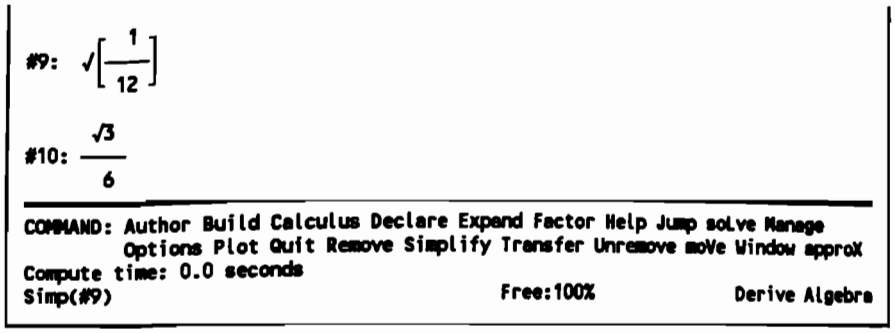
  simplifies to 8.

- SQRT returns the square root of its argument. Note that SQRT(x) is equivalent to $x^{1/2}$. On PC compatible computers SQRT displays as √ and can be entered by pressing **Alt-Q**. For example,

  ```
  √(4 + 5)
  ```

  simplifies to 3.

Exercise: Enter and simplify the square root of 1/12. The screen should look similar to:


#9: √[1/12]
#10: √3/6
COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Compute time: 0.0 seconds
Simp(#9)
Free:100%
Derive Algebra

Note that:

- When entering the expression, the 1/12 must be enclosed in parentheses; and
- The result is simplified and rationalized, but not approximated. Section 3.8 describes how to approximate irrational numbers.

Powers of functions can be entered by putting the power immediately after the function name. For example, entering

```
SQRT^3 25
```

is equivalent to entering

```
(SQRT 25)^3
```

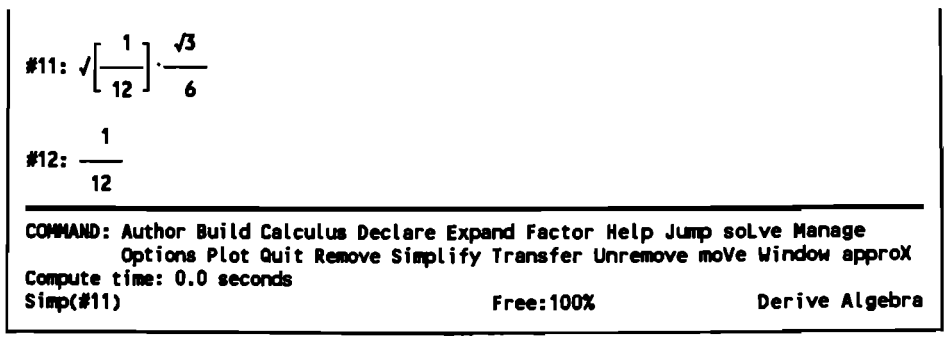
3.5 Building Expressions

Use the **Build** command to combine expressions using operators and functions. This command is similar to the way a pocket calculator is used to perform a sequence of operations on numbers. First you specify an operand, and then you specify an operator. Next you specify an operand, and so on.

After issuing the **Build** command, highlight and/or enter the first expression or subexpression that you want (see Section 3.3) and then press **Enter**. Next you are asked to choose from a list of operators and functions to apply to the expression. This list includes the basic mathematical operators (see Section 3.1) and many of the common mathematical functions (see Chapter 6).

Continue entering expressions and choosing operators until the desired expression is built. Finally, choose the Done option to display the expression in the algebra window. The annotation for the new expression displayed at the left end of the status line will indicate how the expression was built.

Exercise: Build the product of the two expressions from the previous exercise. Then simplify the result. The screen should look similar to:


#11: √[1/12] · √3/6
#12: 1/12
COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Compute time: 0.0 seconds
Simp(#11)
Free:100%
Derive Algebra

Note that:

- When expression #11 is highlighted, the annotation 9*10 indicates that expression #11 is the product of expressions #9 and #10; and
- The simplified result (1/12) is exact with no round-off error.

As the previous exercise shows, the **Simplify** command can be used after building expressions. However, if you have no interest in seeing the unsimplified version of the expression, you can build and simplify an expression in a single step. Just press **Ctrl-Enter** or **Ctrl-J** instead of **Enter** when choosing the Done option.

The **Build** command is useful for extracting and/or combining subexpressions from existing expressions in the algebra window, especially if the subexpressions are too large to fit on the author line. For example, to extract a large subexpression from an expression, highlight the subexpression, issue a **Build** comand, and then select the Done option. The **Build** command is also useful for manually rearranging the terms of a sum or the factors of a product.

Existing expressions can also be combined on the author line using expression labels and/or the **F3** and **F4** function keys (see Section 3.3). Using the **Build** command to combine expressions has the advantage of the annotation providing a record of how the new expression was built.

Exercise: Use the **Build** command to extract $\sqrt{3}$ from the expression in the previous exercise.

3.6 Moving Expressions

Use the **moVe** command to rearrange the order of the expressions in the window. For example, you may want several important expressions to be adjacent so that they are visible simultaneously. Type the letter V to issue this command since another menu option starts with the letter M. The **moVe** command displays a line similar to

MOVE: Before: 13 Start: 13 End: 13

The three selection fields enable you to specify a block of expressions and where to move the block. Use the **Tab** key to move between the fields. Press **Esc** to abort a **Move** command. The block of expressions is moved to its new location when you press **Enter**.

The label numbers in the Start and End fields delimit the block of expressions to move. Note that an entire expression is moved even if only a subexpression is highlighted. To move only one expression, use the same label number in both the Start and End fields.

You can either type the label number of an expression or use the direction keys to highlight an expression (see Section 3.3). Using the direction keys is usually less error prone than typing expression labels.

The label number in the Before field specifies the expression *before* which the block is moved. Type end in this field to move a block of expressions *after* the last expression in the algebra window.

Initially the label number of the highlighted expression is displayed in each of the **moVe** command's fields. Thus to move a single expression, highlight the expression, issue a **moVe** command, move the highlight (see Section 3.3) to the desired destination, and press **Enter**.

Exercise: Use the **Move** command to move the last expression in the algebra window before expression #1.

Note that the rearranged expressions retain their original label numbers. You can use the **Manage Renumber** command to put the label numbers back in sequential order (see Section 3.7).

The **moVe** command always moves a physically contiguous block of expressions. Because of earlier moves, this block may not be a numerically contiguous sequence of label numbers.

3.7 Removing, Unremoving, and Renumbering Expressions

The **Transfer Clear Expressions** command removes *all* expressions from the current algebra window (see Section 2.9).

Use the **Remove** command to remove a physically contiguous sequence of expressions. After awhile, your window is likely to become cluttered with expressions that are no longer of interest. Removing such expressions makes it easier to view and access the expressions that are of interest while conserving memory space.

The **Remove** command displays a line similar to

REMOVE: Start: 13 End: 13

The two selection fields enable you to specify a block of expressions to remove. Use the **Tab** key to move between the fields. Press **Esc** to abort a **Remove** command. The block of expressions is removed when you press **Enter**.

The label numbers in the two fields delimit the block of expressions to remove. Note that an entire expression is removed even if only a subexpression is highlighted. To remove only one expression, use the same label number in both fields.

You can either type the label number of an expression or use the direction keys to highlight an expression (see Section 3.3). Using the direction keys is usually less error prone than typing expression labels.

The **Remove** command always removes a physically contiguous block of expressions. Because of earlier moves, this block may not be a numerically contiguous sequence of label numbers.

Exercise: Use a single **Remove** command to remove all the expressions in the algebra window.

Use the **Unremove** command to recover expressions and their accompanying annotations that may have been inadvertently deleted. This command also makes it possible to move expressions from one algebra window to another (see Section 2.13).

If there are expressions already in the algebra window, the **Unremove** command displays a line similar to

UNREMOVE: Before: 5

enabling you to specify the expression *before* which to insert the previously removed expressions. You can either type the label number of an expression or use the direction keys to highlight an expression (see Section 3.3). Type end in this field to

insert the removed expressions *after* the last expression in the algebra window. The expressions are actually inserted when you press **Enter**.

In order to free memory, the buffer storing the previously removed expressions is emptied whenever an expression is simplified or approximated.

> **Warning:** Removed expressions cannot be recovered once a **Simplify**, **approX**, **Expand**, or **Factor** command is issued.

Moving and/or removing expressions can cause the expression label numbers to get out of order. Use the **Manage Renumber** command to put the expression label numbers in an algebra window in sequential order. This command also updates the expression label numbers in annotations provided the numbers are preceded by the # character.

Expressions and annotations are automatically renumbered when a **Transfer Save Derive** command is issued (see Section 2.8).

3.8 Controlling Precision

Internally ***DERIVE*** stores all numbers as integers or reduced ratios of integers (fractions). All numerical operations are performed using rational arithmetic. Either exact, approximate, or mixed precision can be used.

Use the **Options Precision** command to determine and/or change the precision mode and digits of precision. This command displays a line similar to

OPTIONS PRECISION: Mode: Approximate **Exact** Mixed Digits: 6

The current precision mode is the one highlighted in the mode field. The minimum number of significant decimal digits used in approximate or mixed modes is displayed in the digits field.

When a new precision mode is selected, the **Options Precision** command generates an expression of the form

Precision := *mode*

where *mode* is Approximate, Exact, or Mixed. When a new number of digits of precision is entered, the command generates an expression of the form

PrecisionDigits := *digits*

where *digits* is a positive integer. The precision mode and digits of precision can also be changed by entering on the author line expressions of the above form using upper and lower case exactly as shown.

In **exact** mode, irrational numbers are not approximated and as much memory as necessary is used to store rational numbers. Initially, precision is in exact mode. Exact mode is appropriate when you want an exact answer and must be sure that you have an exact answer.

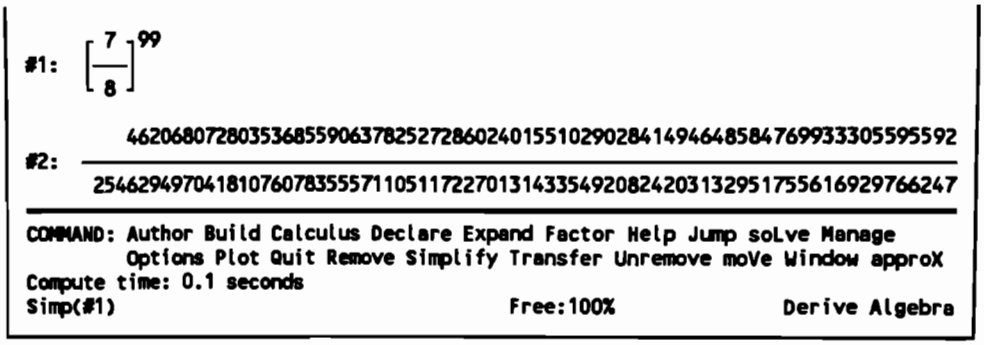
Irrational numbers can not be represented exactly as ratios of integers. Thus, in exact mode, radicals (surds) are left in symbolic form when simplified rather than being approximated. For example, $\sqrt{4}$ simplifies to 2, but $\sqrt{3}$ remains symbolic. In exact mode various techniques are used to simplify expressions involving nested radicals. For example,

$$\sqrt{(4 - 2 \cdot \sqrt{3})}$$

simplifies to

$$\sqrt{3} - 1$$

Exercise: Enter and simplify the expression $(7/8)^{99}$ while in exact mode. The screen should look similar to:


#1: [7/8]^99
#2: 462068072803536855906378252728602401551029028414946485847699333055955592 / 2546294970418107607835557110511722701314335492082420313295175561692976624 7
COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Compute time: 0.1 seconds
Simp(#1)
Free:100%
Derive Algebra

Note that:

- The result is computed exactly in about 0.1 seconds and *stored* as a reduced ratio of two very large integers.
- Part of the result extends off the right end of the screen. Use **Ctrl-Right arrow** or **Ctrl-F** to scroll the expression horizontally to the left (see Section 3.3).

In **approximate** mode, irrational numbers and large rational numbers simplify to the simplest rational number that approximates the original number accurate to the current precision.

The number of significant digits of precision can be set by the **Options Precision** command to any desired positive integer. However, the time required to perform rational operations roughly quadruples when the number of digits of precision is doubled. In addition, huge rational numbers will quickly exhaust available memory.

Approximate mode can be significantly faster and require less memory than exact or mixed mode when rational operations result in numbers having large numerators and denominators.

Approximate mode is appropriate when you want to save time and space at the expense of some roundoff errors. This may be acceptable if the expected roundoff error is negligible compared to errors in experimental data or compared to the accuracy required of the result.

Exercise: Use the **Options Precision** command to change precision to approximate mode, enter $\sqrt{3}$, and then simplify it. Then using another **Options Precision** command, increase the precision to 10 digits and resimplify $\sqrt{3}$. Finally increase the precision to 100 digits and resimplify $\sqrt{3}$. The screen should look similar to:

```
#3:  Precision := Approximate

#4:  √3

#5:  1.73205

#6:  PrecisionDigits := 10

#7:  1.732050807

#8:  PrecisionDigits := 100

#9:  1.732050807568877293527446341505872366942805253810380628055806979451933016 9
──────────────────────────────────────────────────────────────────────────────
COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
         Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Compute time: 0.1 seconds
Simp(#4)                                Free:100%              Derive Algebra
```

Note that:

- Expressions generated by the **Options Precision** command provide a record of the digits of precision used during the approximations.
- Approximate results are displayed in decimal notation.
- If your second approximation is 1.732050**657** instead of 1.732050**807**, you simplified 1.73205 instead of $\sqrt{3}$. Note that 1.73205 is only accurate to 6 digits. To obtain $\sqrt{3}$ accurate to 10 digits, use the direction keys to highlight $\sqrt{3}$, and then issue a **Simplify** command.
- The 100 digit approximation required slightly more time to compute than the 10 digit approximation.

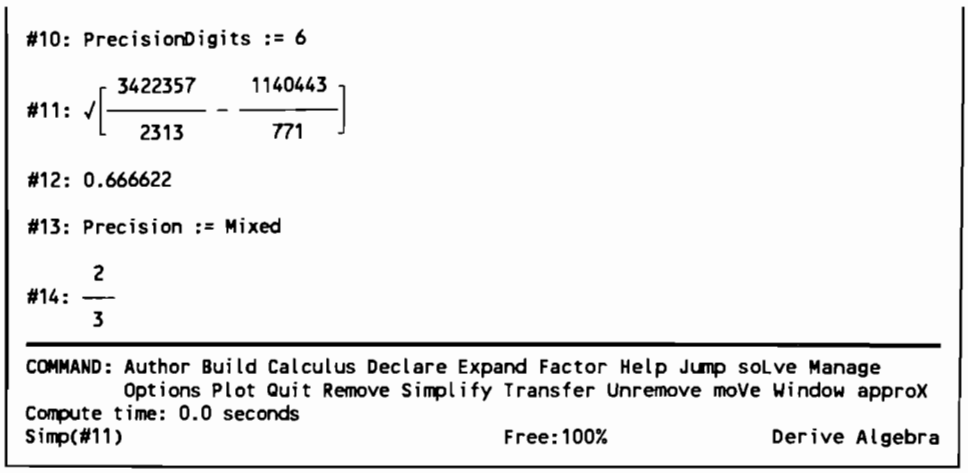
In **mixed** mode, irrational numbers are approximated but rational numbers are not. As with approximate mode, irrational numbers simplify to the simplest rational number that approximates the original number accurate to the current precision.

Mixed mode is appropriate when you are willing to approximate irrational operations to force numeric results but want all rational operations done exactly.

Exercise: Use the **Options Precision** command to select approximate mode with 6 digits of precision. Next, enter and simplify

$$\sqrt{(3422357/2313 - 1140443/771)}$$

Then use the **Options Precision** command to change precision to mixed mode. Finally highlight and simplify the above expression. The screen should look similar to:


#10: PrecisionDigits := 6
#11: √[3422357/2313 - 1140443/771]
#12: 0.666622
#13: Precision := Mixed
#14: 2/3
COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Compute time: 0.0 seconds
Simp(#11)
Free:100%
Derive Algebra

Note that:

- Approximate mode gives an approximate result because rounding the rational numbers *before* they are subtracted results in significant roundoff error.
- Mixed mode gives an exact result because the difference of the two rational numbers is computed exactly.

Regardless of the number in the digits field, exact mode uses as many digits as necessary for exactness. Thus, set the digits field to the precision you want for approximate calculations.

If you change the precision and issue another **Options Precision** command, the number of digits may exceed by one the precision you requested. ***DERIVE*** stores rational numbers using enough bytes of memory to provide the current precision. Since each byte provides more than one decimal digit of precision, the precision may be slightly more than you requested.

Most individual roundoff errors are a few digits more accurate than the indicated precision. However, roundoff errors may accumulate in a sequence of operations.

Moreover, subtracting two nearly equal approximate numbers may yield a result having few or no significant digits. This is called **catastrophic cancellation**.

For example, if n is a large positive integer, a difference of the form

$$\sqrt{(n^2 + 1)} - n$$

leads to catastrophic cancellation since $\sqrt{(n^2 + 1)}$ and n approximate to nearly equal numbers. Thus, in approximate mode with 6 digits of precision

$$\sqrt{10001} - 100$$

simplifies to 0.00499722. However, only the first three digits of this result are correct. This can be verified by increasing the digits of precision to 10 and resimplifying the above difference.

As opposed to the floating-point arithmetic used by most calculators and mathematical software, approximate rational arithmetic has the useful property that simple exact rational results are often obtained despite intermediate roundings or irrational approximations.

If an irrational expression is equivalent to a simple fraction, then approximate mode is more likely than mixed mode to yield that simple fraction. Otherwise, mixed mode is likely to be more accurate.

Exercise: Simplify the expression

$$\frac{\sqrt{(\sqrt{3} + 1)} + \sqrt{(3 \cdot \sqrt{3} + 3)}}{\sqrt{(24 \cdot \sqrt{3} + 40)}}$$

in exact, approximate, and mixed mode with 6 digits of precision. Note that both the exact result (1/2) and the approximate result (0.5) are exact; whereas, the mixed mode result (0.499999) is close but only approximate.

In exact mode ***DERIVE*** tries to denest expressions involving nested radicals. For example,

$$\sqrt{(5 + 2 \cdot \sqrt{6})}$$

simplifies to $\sqrt{3} + \sqrt{2}$, and

$$(243 \cdot \sqrt{5} - 294 \cdot \sqrt{3})^{1/3}$$

simplifies to $3\sqrt{5} - 2\sqrt{3}$. However, it is difficult if not impossible to fully simplify all such expressions involving nested radicals (but we're working on it!).

3.9 Controlling Notation

Precision is the accuracy used to *compute* numbers (see Section 3.8). A related but separate issue is the notation used to *display* numbers.

Numbers can be displayed using **rational notation**: Integers are displayed as integers and fractions are displayed as ratios of integers. Rational notation has the advantage that all stored numbers can be displayed exactly. However, numbers that require many digits to display in rational notation are difficult to comprehend.

Thus, it is also possible to display fractions using **decimal point notation**. It is also possible to display large magnitude and small magnitude numbers using **scientific notation**. Finally, it is possible to display numbers in **mixed notation** using whichever style is most concise: rational notation if a number is simple, decimal point notation if a number's magnitude is not extreme. Otherwise use scientific notation.

You can also control the maximum number of fractional digits displayed in decimal point notation or the maximum number of significant digits displayed in scientific notation. This number can be more, less, or the same as the number of digits of precision set by a **Options Precision** command. For decimal fraction, scientific, or mixed notation, it is usually advisable to display no more digits than are accurate and necessary for your purposes. This policy avoids misleading results and increases comprehension.

Use the **Options Notation** command to see and/or change the notation style and the number of digits used when displaying numbers on the screen *and* when saving numbers in MTH files. This command displays a selection field line similar to

OPTIONS NOTATION: Style: Decimal Mixed **Rational** Scientific Digits: 6

Regardless of the number in the digits field, rational notation uses as many digits as necessary to display results. Thus, set the digits field to the digits you want displayed for decimal fraction, scientific, or mixed notation.

When a new notation style is selected, the **Options Notation** command generates an expression of the form

Notation := *style*

where *style* is Decimal, Mixed, Rational, or Scientific. When a new number of digits to display is entered, the command generates an expression of the form

NotationDigits := *digits*

where *digits* is a positive integer. The notation style and digits to display can also be changed by entering on the author line expressions of the above form using upper and lower case exactly as shown.

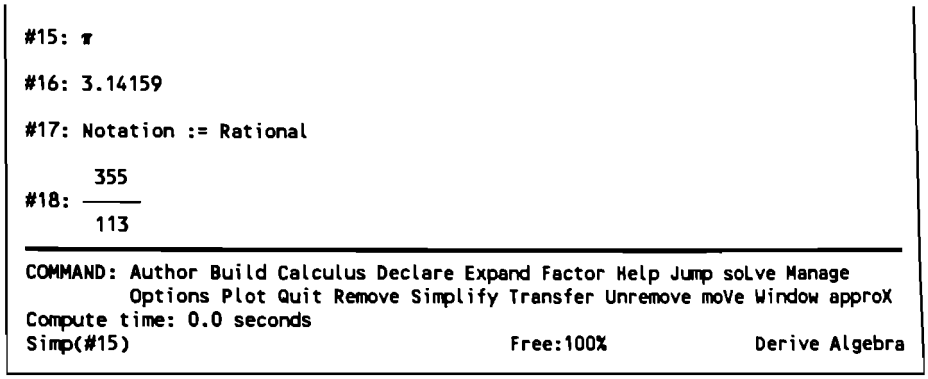
If you use the **Options Precision** command to change the precision to exact mode, the notation style is automatically changed to rational. If you change the precision to approximate mode, the notation style is changed to scientific. If you change the precision to mixed mode, the notation style is changed to mixed.

If you use the **Options Precision** command to change the number of digits of precision, the number of digits displayed is changed to the *requested* precision.

Normally these automatic changes made to the notation by **Options Precision** commands make **Options Notation** commands unnecessary. However, if you want some other notation, the **Options Notation** command must be issued *after* the **Options Precision** command.

π is the ratio of the circumference to diameter of a circle. It can be entered by typing pi. On PC compatible computers it can also be entered by pressing **Alt-P**.

Exercise: Use an **Options Precision** command to set precision to approximate mode using 6 significant digits, then enter and simplify π. Next use an **Options Notation** command to set the notation style to rational, and resimplify π. The screen should look similar to:



Note that:

- The first result is a *decimal* approximation of π.
- The second result is a *rational* approximation of π.

3.10 Approximating Irrational Numbers

Normally it is best and safest to run *DERIVE* using exact mode arithmetic. In exact mode you may obtain expressions containing subexpressions that do not simplify to rational numbers. Use the **approX** command or the APPROX function to approximate the irrationals in such expressions without having to change the precision back-and-forth.

Simplifying expressions using the **approX** command is the same as simplifying expressions using the **Simplify** command except that the precision is temporarily set to approximate and the notation style is temporarily set to scientific. The precision used is that indicated by the Digits field of the **Options Precision** command (see Section 3.8). The number of digits displayed is that indicated by the Digits field of the **Options Notation** command (see Section 3.9).

The APPROX function can be used on the author line to approximate expressions. APPROX(u,n) approximates expression u using n digits of precision. n defaults to that indicated by the Digits field of the **Options Precision** command.

Exercise: Use an **Options Precision** command to set precision to exact mode. Highlight and simplify π and note that it is *not* approximated. Then use an **approX** command to approximate π.

Exercise: Using the default precision of 6 significant digits, approximate the expression

$$e^{\pi \cdot \sqrt{163}}$$

Noting that the result has 18 whole digits, inspect the first 7 digits of the fractional part as follows: After setting the precision to 25 digits and the notation to decimal, reapproximate the *original* expression and note that the result is an integer!

Is this a remarkable identity that these three irrational numbers e, π, $\sqrt{163}$ combine to give an integer? To gain additional evidence, keep increasing the precision and approximating the original expression until the result is clearly not an integer, or until you become convinced that it actually is an integer and can prove it.

3.11 Controlling Radix Base

You may have a need to convert numbers between radix bases or to work with numbers in bases other than base ten. The most commonly used radix bases are binary (base two), octal (base eight), decimal (base ten), and hexadecimal (base sixteen). Skip this section if you do not need to work with numbers in bases other than base ten.

You can independently control the base used for entering numbers and the base used for displaying numbers. Use the **Options Radix** command to view and/or change the current input base and output base. It displays two field lines similar to

OPTIONS RADIX: Input: Binary Octal **Decimal** Hexadecimal Other
Output: Binary Octal(Decimal) Hexadecimal Other

where the current input base is highlighted and the current output base is enclosed in parentheses.

If you select Other, you will be prompted to enter an integer between 2 through 36 inclusive. The current radix will be displayed in decimal (base ten), and your choice of radix must be entered in decimal (base ten).

When a new input radix base is selected, the **Options Radix** command generates an expression of the form

InputBase := *base*

where *base* is Binary, Octal, Decimal, Hexadecimal, or an integer between 2 and 36 inclusive. When a new output radix base is selected, the command generates an expression of the form

OutputBase := *base*

The input and output radix bases can also be changed by entering on the author line expressions of the above form using upper and lower case exactly as shown.

For radix bases greater than ten, the letter A is used to represent the digit ten, the letter B is used to represent the digit eleven, and so on through the letter Z for the digit thirty-five.

To avoid ambiguity with variables, such numbers that start with a letter are displayed and must be entered using a leading zero (0) digit. For example, in base sixteen the decimal number fourteen is displayed and must be entered as 0E.

If you enter a number and it displays differently than what you typed, you may have inadvertently changed the input and/or output radix. Use an **Options Radix** command or a **Help State** command to determine the current radix.

Exercise: Use an **Options Radix** command to set the output radix to hexadecimal (base sixteen) while leaving the input radix decimal (base ten). Then try entering some numbers in decimal and note that they are displayed in hexadecimal. To avoid confusion, restore the output radix to decimal (base ten) before proceeding to the next section.

3.12 Annotating Expressions

A good way to document what transpired during a ***DERIVE*** session is to record how the expressions were derived. When an expression is authored, simplified, approximated, or built, ***DERIVE*** automatically annotates the expression with a brief indication of how it was derived (see Sections 3.1, 3.2, 3.5, and 3.10).

The **Transfer Save Derive** command (see Section 2.8) normally saves annotations along with the corresponding expressions in MTH files so they can be restored by the **Transfer Load Derive** command. Also when a **Transfer Print Expressions** command is issued, annotations are normally printed with the corresponding expressions.

Use the **Manage Annotate** command to modify or expand upon the annotation for an expression. This command displays a line similar to

MANAGE ANNOTATE: Expression: 10

After you to enter an expression by typing an expression label number or using the arrow keys, the expression's current annotation is displayed. You can edit the annotation as desired (see Section 2.4). Press **Enter** when you are satisfied with your new annotation. Press **Esc** to abort the **Manage Annotate** command without changing the annotation.

Annotations can be as long as desired, but only the first 40 characters are displayed on the algebra window's status line. However, entire annotations are sent to the printer by the **Transfer Print Expressions** command.

Exercise: Appropriately annotate the expressions resulting from the exercises in Section 3.10. Then print out the expressions and annotations.

DERIVE comes with a number of DMO files that demonstrate the capabilites of the system (see Section 1.1). The **Manage Annotate** command makes it possible to create your own demonstration files. First, enter the expressions to be simplified during the demonstration. Next, annotate the expressions with the comment to be displayed when the demonstration is run. Finally, save the expressions and annotations in a file whose type name is DMO using the **Transfer Save Derive** command.

3.13 Controlling Expression Format

Use the **Options Output** command to control how expressions are displayed on the screen or sent to the printer.

Use the Format field of the **Options Output** command to select Normal or Compressed display format for expressions. In normal display format, spaces are used around some operators, like + and −, to enhance the readability of expressions. In compressed display format, nonessential spaces around operators are eliminated. Compressed format may make it possible to display a large expression without having to use horizontal scrolling. The display format setting is saved in *DERIVE* initialization files (see Section 2.14). The initial display format setting is Normal format.

The proper symbol to use for the times operator, used to indicate multiplication, is a subject of heated debate and a matter of personal taste. Therefore, *DERIVE* gives you a choice. Use the Operator field of the **Options Output** command to select the Asterisk, Dot, or Implicit to indicate multiplication. If Implicit is selected, a space is inserted between operands to indicate multiplication. The times operator is saved in *DERIVE* initialization files (see Section 2.14). The initial times operator is Dot.

When a new display format is selected, the **Options Output** command generates an expression of the form

DisplayFormat := *format*

where *format* is Normal or Compressed. When a new times operator is selected, the command generates an expression of the form

TimesOperator := *operator*

where *operator* is Asterisk, Dot, or Implicit. The display format and times operator can also be changed by entering on the author line expressions of the above form using upper and lower case exactly as shown.

After selecting a new display format and/or times operator, an existing expression can be reformatted by highlighting a subexpression of the expression using the direction keys (see Section 3.3).

Note that selecting both Compressed display format and the Implicit times operator results in poorly spaced expressions.

Exercise: Examine the expression

`a·b + c·b`

using each of the six possible settings provided by the **Options Output** command.

Chapter 4. Algebra

Most mathematical computer programs can substitute numeric values into formulas. A few programs can solve equations approximately, thus obtaining numeric solutions. What makes ***DERIVE*** unique is that it is also able to combine and simplify *formulas*. For example, it can algebraically as well as numerically solve an equation for one variable in terms of the other variables.

This chapter describes how to enter, simplify, and solve algebraic expressions. Also described are commands for ordering variables, factoring and expanding expressions, substituting values for variables, defining functions, declaring variable domains.

For a demonstration of these capabilities, load the file ALGEBRA.DMO using a **Transfer Demo** command (see Section 2.9).

4.1 Entering Variables

Expressions in ***DERIVE*** can contain variables whose values are unknown. The letters of the Latin (English) alphabet can be used as variable names.

Some letters of the Greek alphabet can also be used as variable names. A Greek letter is entered by typing the equivalent English word. On PC compatible computers, a Greek letter can also be entered by holding down the **Alt** key while pressing a letter key. The following table summarizes the available Greek letters and the corresponding Latin words and **Alt** keys:

Greek letter	English word	Alt key	Greek letter	English word	Alt key
α	alpha	**Alt-A**	μ	mu	**Alt-M**
β	beta	**Alt-B**	π	pi	**Alt-P**
Γ	gamma	**Alt-G**	σ	sigma	**Alt-S**
δ	delta	**Alt-D**	τ	tau	**Alt-T**
ϵ	epsilon	**Alt-N**	Φ	phi	**Alt-F**
θ	theta	**Alt-H**	Ω	omega	**Alt-O**

The names of the pre-defined functions and constants (see Chapter 6) are reserved names. Thus, for example, ABS, SQRT, pi, π, GAMMA, and Γ, can *not* be used as variable names.

On PC compatible computers, characters in the extended ASCII character set (see Section 2.10) can be entered by holding down the **Alt** key and while typing the integer character codes. Your computer manuals should list the character codes.

Use the **Options Input** command to view and/or change the current input, case, and arrow key modes. This command displays three selection fields:

OPTIONS INPUT: Mode: Character Word Case: Insensitive Sensitive
Use: LineEdit Subexpression

When a new variable input mode is selected, the **Options Input** command generates an expression of the form

InputMode := *mode*

where *mode* is Character or Word. When a new variable case mode is selected, the command generates an expression of the form

CaseMode := *mode*

where *mode* is Insensitive or Sensitive. When a new arrow key mode is selected, the command generates an expression of the form

ArrowKeyMode := *mode*

where *mode* is LineEdit or Subexpression. The input, case, and arrow key modes can also be changed by entering on the author line expressions of the above form using upper and lower case exactly as shown.

Declared variables are variables that have been assigned a domain (see Section 4.10) or a value (see Section 4.11). Declared variables can have single or multi-character names. Use the Mode field of the **Options Input** command to select the input mode for the names of *un*declared variables:

- In **character input mode**, undeclared variables can have names consisting of only a single Latin or Greek letter (except as noted below). If you are satisfied with such names, this input mode eliminates having to insert a space between variables being multiplied. For example,

  xyz

  is interpreted as the product of the variables x, y, and z. Character mode is the initial variable input mode.

- In **word input mode**, undeclared variables can have names consisting of any number of letters, digits, or underscore characters, provided they begin with a letter. For example, in word input mode, x1, velocity, and light_years are valid multi-character variable names. Note that word input mode requires that undeclared variables be separated by a space, an operator, a function, or a parenthesis.

In either input mode, undeclared variables can have names consisting of a single letter followed by one or more underscore characters, or of an at-sign character followed by one or more decimal digits. For example, in character or word input mode, x_ and @123 are valid variable names.

In either input mode, undeclared variables can have multi-character names consisting of any character, including spaces and mathematical operators, provided the name is enclosed in double quote marks. This provides a convenient way to include comments in ***DERIVE*** sessions that are saved in MTH files. For example, the following is a valid variable name:

```
"If x < 3, the following equation holds true:"
```

Use the Case field of the **Options Input** command to select case-sensitive or case-insensitive mode for variable and function names:

- In **case-insensitive mode**, variable and function names can be entered in *either* upper-case or lower-case letters. In this mode variable names are displayed in lower-case letters and function names are displayed in upper-case letters. Case-insensitive is the initial mode.
- In **case-sensitive mode**, case can be used to distinguish variable names and function names. Variable and function names are displayed in the case in which they are entered. For example, pwr, Pwr, and PWR can be used as the names of three distinct variables and/or functions. Note that in this mode, the pre-defined functions (see Chapter 6) must be typed in upper-case letters.

Use the Use field of the **Options Input** command to select the initial arrow key mode on the author line:

- If **line-edit mode** is selected, when an **Author** command is issued, the **Left arrow**, **Right arrow**, **Home**, and **End** keys move the cursor on the author line (see Section 2.4). Initially line-edit is the initial arrow key mode.
- If **subexpression mode** is selected, initially these keys move the highlight within an expression (see Section 3.3).

In either initial arrow key mode, pressing **F6** toggles between line-edit and subexpression modes when entering an expression on the author line (see Section 3.3).

All the **Options Input** command mode settings are saved in ***DERIVE*** state initialization (INI) files, so you can change the initial settings as desired (see Section 2.14).

4.2 Simplifying Expressions

Use the **Simplify** command to simplify mathematical expressions. The following are some of the simplifications this command performs:

- combining numeric subexpressions, as in

$$2 \cdot y \cdot 3 \rightarrow 6 \cdot y$$

- collecting similar factors of a product, as in

$$x^2 \cdot y \cdot x \rightarrow x^3 \cdot y$$

- collecting similar terms of a sum, as in

$$3 \cdot x + 7 + x \rightarrow 4 \cdot x + 7$$

- employing identities involving 0, as in

$$x + 0 \rightarrow x$$

- employing identities involving 1, as in

$$1 \cdot x \rightarrow x$$

- distributing integer exponents over a product, as in

$$(3 \cdot x \cdot y^3)^2 \rightarrow 9 \cdot x^2 \cdot y^6$$

- canceling polynomial greatest common divisors, as in

$$\frac{x^2 + 2 \cdot x \cdot y + y^2}{x^2 - y^2} \rightarrow \frac{x + y}{x - y}$$

- expanding products and integer powers of polynomials to the extent that it may eliminate a variable or reduce the degree in a variable, as in

$$(x + 1)^2 - x^2 \rightarrow 2 \cdot x + 1$$

- forcing common denominators to the extent that they may eliminate a variable or reduce the degree in a variable, as in

$$\frac{2 \cdot x}{x^2 - 1} - \frac{1}{x - 1} \rightarrow \frac{1}{x + 1}$$

Some of the classes of mathematical expressions that ***DERIVE*** can simplify are:

- **Univariate polynomials** composed of one variable, rational numbers, addition, subtraction, multiplication, and integer powers.
- **Multivariate polynomials** similarly composed but having more than one variable.
- **Rational expressions** composed of polynomials and ratios of polynomials.
- **Extended rational expressions** composed of rational expressions and fractional powers of variables.
- **Algebraic expressions** composed of rational expressions and fractional powers of rational and algebraic expressions.
- **Elementary transcendental expressions** composed of algebraic expressions and elementary functions (exponential, logarithmic, trigonometric, hyperbolic, and their inverses).

A **sufficiently simple expression** has no superfluous variables, roots, or functions, and no reducible polynomial degrees. Insufficiently simple expressions mislead you about the dependence and qualitative behavior of the expression. For example, a result would be highly misleading if it appeared to be quadratic in a variable when it was actually linear.

The primary goal of the **Simplify** command is to transform expressions into a sufficiently simple form. To accomplish this, it converts expressions to a **normal form** that automatically reduces most expressions to a sufficiently simple form.

Beyond the class of extended rational expressions, it is impractical and often impossible to guarantee a sufficiently simple form. The **Simplify** command tries hard nonetheless, and often succeeds even for these more complicated expressions.

The secondary goal of the **Simplify** command is to transform expressions as little as necessary to achieve the primary goal. In support of this, the normal form is as flexible as practical while still ensuring a sufficiently simple form for most expressions. Thus the normal form often permits expressions to remain unexpanded, unfactored, or without a common denominator. For example,

$$x^2 - (x + (y + 1)^{50})\,(x - (y + 1)^{50})$$

simplifies to

$$(y + 1)^{100}$$

without the result being expanded to the 100th power.

Despite its flexibility, the normal form can force transformations that do not yield a class simplification. Therefore, ***DERIVE*** makes it easy to simplify just part of an expression. When the **Simplify** command prompts you to select an expression, highlight just the subexpression that you want simplified (see Section 3.3). The result is a copy of the whole expression in which *only* the highlighted portion is simplified.

Exercise: Enter and simplify

$$(5 \cdot x - 3 \cdot x + 1)^7 - x$$

Then highlight and simplify just the subexpression enclosed in parentheses. The screen should look similar to:

```
                    7
#1:  (5·x - 3·x + 1)  - x

          7        6        5        4        3        2
#2:  128·x  + 448·x  + 672·x  + 560·x  + 280·x  + 84·x  + 13·x  + 1

              7
#3:  (2·x + 1)  - x

COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
         Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Compute time: 0.0 seconds
Simp(#1')                              Free:100%              Derive Algebra
```

Note that:

- Simplifying the entire expression results in an expression that does not annihilate a variable or reduce the degree.
- Simplifying just the subexpression results in a simpler, more compact form of the expression.
- The single quote mark in the derivation of expression #3 indicates that only part of expression #1 was simplified.

If all you want to do is build and/or edit an expression, you do not need to use the **Simplify** command at all. Just use a **Build** command (see Sections 3.5) to construct a new expression. Use an **Author** command (see Section 3.1) in conjunction with the **F3** function key (see Section 3.3) to edit an existing expression.

Depending on the expression, it can take a long time to do the expansions and canceling of common divisors necessary to simplify an expression. Press **Esc** to abort a **Simplify** command that is taking too long. You can then try simplifying only portions of the expression. For example, you can separately simplify complicated numerators and denominators. The resulting expression may be incompletely simplified, but it is better than no simplification at all.

Simplification can also require a large amount of computer memory to store intermediate results. If space is exhausted while simplifying an expression, the warning message

Memory Full

is displayed on the message line, an error beep is sounded, and no result is returned.

If the **Simplify** command generates a large expression that is not of interest, use the **Remove** command (see Section 3.7) to save computer memory and screen space.

4.3 Ordering Variables

Since addition and multiplication are commutative and associative, the terms of a sum and the factors of a product could be arranged in any order. However, the *DERIVE* normal form orders terms and factors in a certain way to aid the collection of similar terms and factors. For example, the similar terms of

$$5 \cdot y \cdot x + 7 - 3 \cdot x \cdot y$$

are much easier to find and collect if the terms are first rearranged as

$$5 \cdot x \cdot y - 3 \cdot x \cdot y + 7$$

The **Manage Ordering** command lets you control the ordering used to sort variables. This variable ordering has a strong influence on how the terms of a sum and the factors of a product are ordered.

DERIVE maintains a list of variables that specifies the ordering used to sort variables. This variable **order list** initially consists of the three variables x, y, and z.

A variable on the list is *more* main than variables after it on the list or variables not on the list. A variable not on the list is *less* main than variables on the list or variables before it in alphabetical order. For example, x is more main than y, which is more main than a.

When an expression has more than one variable, the most main variable is called the **primary variable**. The next most main variable is called the **secondary variable**, and so on. For example, x is the primary variable of

$$5 \cdot a \cdot x^2 + x \cdot y$$

y is the secondary variable, and a is the tertiary variable.

The **coefficient** of a product consists of those factors that do *not* contain the primary variable of the product. For example, if x is the primary variable of

$$3 \cdot a \cdot x^2 \cdot y^5$$

then $3ay^5$ is the coefficient, and 3 is the **numeric coefficient**.

The ordering of the factors in a product is based solely on the bases of the factors; the exponents of the factors have *no* influence on the ordering. Bases of a product that consist of a variable on the order list are sorted in that order. For example,

$$z^2 \cdot x^3 \cdot y^5$$

simplifies to

$$x^3 \cdot y^5 \cdot z^2$$

Bases of a product that consist of a variable *not* on the order list are sorted in alphabetical order *before* bases that consist of a variable on the order list and *after* the numeric coefficient of the product. For example,

$$z^3 \cdot b^2 \cdot a \cdot 2 \cdot x^5 \cdot c$$

simplifies to

$$2 \cdot a \cdot b^2 \cdot c \cdot x^5 \cdot z^3$$

When ordering the terms of a sum, terms that contain the primary variable come before terms free of the variable. Moreover, terms that contain the primary variable come before terms that contain that variable to a lower degree. For example,

$$5 \cdot y^3 + 2 \cdot x^2 - 3 \cdot a \cdot x$$

simplifies to

$$2 \cdot x^2 - 3 \cdot a \cdot x + 5 \cdot y^3$$

The coefficients of terms are used to break ties in the ordering of sums. The terms of the sum that do not contain the primary variable of the sum are ordered using their own primary variable.

DERIVE sometimes rearranges the terms of a sum to produce more attractive output by eliminating a leading minus sign or displaying the real part of a sum before the imaginary part. For example,

$$-x + y$$

simplifies to

$$y - x$$

Use the **Manage Ordering** command to change and/or rearrange the variables on the variable order list. This command displays the current list and invites you to make changes. Note that existing expressions must be resimplified to see the effect of changing the variable order list.

If changing the variable order list does not yield exactly the desired form for a simplified expression, you can rearrange the expression using a **Build** or **Author** command.

Subexpressions are less likely to be expanded if their variables are least main. Similarly, ratios are least likely to be combined if variables in sums in their denominators are least main. Thus the ordering of variables can dramatically affect the conciseness of a simplified result and the time required to obtain it.

Exercise: Enter and simplify

$$(x + 1)^9 + y$$

Then use a **Manage Ordering** command to change the order list to: $y\ x\ z$, and resimplify the expression. Finally, restore the order list to the initial: $x\ y\ z$. The screen should look similar to:

```
            9
#1:  (x + 1)  + y

     9      8       7       6       5       4       3       2
#2:  x  + 9·x  + 36·x  + 84·x  + 126·x  + 126·x  + 84·x  + 36·x  + 9·x + y + 1

              9
#3:  y + (x + 1)

COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
         Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Compute time: 0.0 seconds
Simp(#1)                                Free:100%              Derive Algebra
```

Note that:

- When x is more main than y (as it is initially), the subexpression $(x+1)^9$ is expanded, thus giving a bulky result.
- When y is more main than x, the subexpression $(x+1)^9$ is *not* expanded, thus giving a compact result that requires less time to compute.

Expression growth from an unfortunate ordering of variables can exhaust memory before a result is derived. If this occurs while simplifying a multivariate expression, change the ordering of the variables and try resimplifying the expression. Alternatively, use a **Manage Substitute** command (see Section 4.8) to interchange the names of the variables in the expression before resimplifying.

4.4 Expanding Polynomials

The general objective when expanding an expression with respect to designated variables is to maximize the number of terms that are algebraically independent with respect to those variables.

Use the **Expand** command or the EXPAND function to expand an expression or subexpression with respect to some or all of its variables. If a polynomial expression is expanded, the polynomial expansion is returned. If a rational expression is expanded, the partial fraction expansion is returned (see Section 4.7). Note that the **Expand** command and the EXPAND function do *not* control exponential, logarithmic, and trigonometric expansion and collection. For that, see Chapter 6.

The **Expand** command prompts you to select an expression or subexpression to expand. If you highlight a subexpression, the result is a copy of the whole expression in which only the subexpression is expanded. For example, if you highlight the whole expression

$$2 \cdot x \cdot (x - 3)^2$$

it expands to

$$2 \cdot x^3 - 12 \cdot x^2 + 18 \cdot x$$

whereas if you highlight just the subexpression $(x-3)^2$, the expression expands to

$$2 \cdot x \cdot (x^2 - 6 \cdot x + 9)$$

An expression that contains more than one variable can be expanded with respect to some or all of its variables. The **Expand** command displays a list of the variables on the message line, and you are prompted to enter the desired **expansion variables**. Press **Enter** if you want expansion with respect to all of the variables in the displayed order. Otherwise enter the desired primary expansion variable, the secondary expansion variable, etc. These expansion variables temporarily override the ordering specified by the variable order list (see Section 4.3).

The EXPAND function can be used on the author line to expand an expression when it is simplified. The call EXPAND(u, x, y, ...) expands expression $u(x, y, ...)$ with respect to the variables x, y, ... If no variables are given, expansion is done with respect to all the variables in the expression.

Terms having the same degrees for all expansion variables are collected. Non-expansion variables that occur in these collected terms are not expanded unnecessarily. For example, if x and y are the expansion variables, the expression

$$3 \cdot a^2 \cdot x^2 \cdot y^3 + 7 \cdot (b + 1)^2 \cdot x^2 \cdot y^3$$

expands to

$$x^2 \cdot y^3 \cdot (3 \cdot a^2 + 7 \cdot (b + 1)^2)$$

To see the effect of changing the expansion variables, consider expanding the expression

$$(x + 2 \cdot y + 1)^3$$

- Expanding with respect to x alone yields

$$x^3 + 3 \cdot x^2 \cdot (2 \cdot y + 1) + 3 \cdot x \cdot (2 \cdot y + 1)^2 + (2 \cdot y + 1)^3$$

- Expanding with respect to y alone yields

$$8 \cdot y^3 + 12 \cdot y^2 \cdot (x + 1) + 6 \cdot y \cdot (x + 1)^2 + (x + 1)^3$$

- Expanding with respect to x as the primary variable and y as the secondary variable yields

$$x^3 + 6 \cdot x^2 y + 3 \cdot x^2 + 12 \cdot x \cdot y^2 + 12 \cdot x \cdot y + 3 \cdot x + 8 \cdot y^3 + 12 \cdot y^2 + 6 \cdot y + 1$$

- Expanding with respect to y as the primary variable and x as the secondary variable yields

$$8 \cdot y^3 + 12 \cdot y^2 x + 12 \cdot y^2 + 6 \cdot y \cdot x^2 + 12 \cdot y \cdot x + 6 \cdot y + x^3 + 3 \cdot x^2 + 3 \cdot x + 1$$

As illustrated by these examples, comprehensibility tends to decline with the number of expansion variables. For most problems you only need to expand with respect to a single variable.

Depending on the expression and the expansion variables, it may take a long time to expand an expression. Press **Esc** to abort an **Expand** command that is taking too long. You can then try expanding only portions of the expression or expanding with respect to a different variable. These actions may drastically reduce the computing time and improve the comprehensibility of a result.

If the **Expand** command generates a large expression that is not of interest, use the **Remove** command (see Section 3.7) to save computer memory and screen space.

Note that both the **Expand** and **Simplify** commands transform an expression into a sufficiently simple form (see Section 4.2). However, the **Simplify** command is often much faster and usually yields an expression more similar to the original expression.

Exercise: Enter and expand with respect to all variables the expressions $(a+b)^2$, $(a+b+c)^2$, $(a+b+c+d)^2$, etc. until you can predict the form of the answer. Repeat the exercise expanding the expressions with respect to just the variable a.

Exercise: Enter and expand the expressions $(x+1)^2$, $(x+1)^3$, $(x+1)^4$, etc. until you can predict the form of the answer.

> **Hint:** Each of the coefficients of the nth degree expansion can easily be computed using two of the coefficients of the $n-1$th degree expansion.

4.5 Choosing Complex Branches

In mathematics the **imaginary unit**, $\sqrt{(-1)}$ is usually denoted by the letter i. To distinguish it from the variable name i, ***DERIVE*** displays the imaginary unit as $\hat{\imath}$ on PC compatible computers and as `#i` on other computers. The exponentiation symbol over the character is to remind you of the exponentiation symbol used when entering the imaginary unit as

```
(-1)^(1/2)
```

The imaginary unit is entered by typing `#i` or by pressing **Alt-I** on PC compatible computers.

A **complex expression** is an expression that contains the imaginary unit. A **real expression** does not contain the imaginary unit. Operations such as factoring and equation solving can yield complex expressions. Therefore, it is important to recognize them even if you do not plan to enter any complex expressions.

A complex expression is in **rectangular form** if it consists of the sum of a real expression and the imaginary unit times another real expression. ***DERIVE*** simplifies complex expressions to rectangular form. For example,

$$(1 + \hat{\imath})^3$$

simplifies to

$$-2 + 2 \cdot \hat{\imath}$$

and

$$\frac{\hat{\imath} + 7}{3 \cdot \hat{\imath} + 4}$$

approximates to

$$1.24 - 0.68 \cdot \hat{\imath}$$

A point in the complex plane is determined by its **magnitude** (i.e. its distance from the origin) and its **phase** (i.e. the angle of the point from the x-axis). By definition, the phase angle in radians is restricted to the interval

$$-\pi < \textit{phase angle} \leq \pi$$

If z is a point in the complex plane and n is an integer, then $z^{1/n}$ could be any one of n distinct points, called **branches**, in the complex plane: The magnitude of $z^{1/n}$ is equal to $|z|^{1/n}$, but the phase of $z^{1/n}$ could be any one of n equally spaced angles. For example, the magnitude of $9^{1/2}$ is 3, but the phase can be either 0 radians yielding 3, or π radians yielding -3.

The **principal branch** of $z^{1/n}$ is the point in the complex plane whose magnitude is $|z|^{1/n}$ and whose phase is the phase of z divided by n. For example, the principal branch of $9^{1/2}$ is 3 and the principal branch of $(-8)^{1/3}$ is $1+\sqrt{3}\,i$.

Use the **Manage Branch** command to change the branch ***DERIVE*** returns when simplifying fractional powers. It displays the selection field

MANAGE BRANCH: **Principal** Real Any

When a new branch option is selected, the **Manage Branch** command generates an expression of the form

Branch := *option*

where *option* is Principal, Real, or Any. The branch option can also be changed by entering on the author line expressions of the above form using upper and lower case exactly as shown.

Select the Principal branch option to force ***DERIVE*** to always select the principal branch. Initially the principal branch is used since it is the most widely accepted systematic choice.

Select the Real branch option to force ***DERIVE*** to select a real branch if one exists, even if it is not the principal branch. For example, if this option is selected, $(-8)^{1/3}$ simplifes to -2 instead of $1+\sqrt{3}\,i$. Note that this branch selection is preferable for most calculus and precalculus applications.

Select the Any branch option to allow ***DERIVE*** to use the maximum number of transformations when it does not matter which branch is choosen. For example, the plus-or-minus "operator" in the quadratic formula constructs both solutions no matter which branch is chosen for the square root in the formula. In Any branch mode, ***DERIVE*** applies the transformations

$$(z^m)^{1/n} \rightarrow z^{m/n}$$

$$(z\,w)^{1/n} \rightarrow z^{1/n}\,w^{1/n}$$

even if the phases of z and w are unknown. For example, in Any branch mode

$$(x^2)^{1/2}$$

simplifies to x instead of the more conservative $|x|$, even if the sign of x is unknown. Thus instead of consistently using a single branch in Any branch mode, ***DERIVE*** uses the positive branch when x is positive and the negative branch when x is negative.

If n and m are integers, $z^{m/n}$ is equivalent to $(z^{1/n})^m$. Thus when simplifying $z^{m/n}$ *DERIVE* first picks the appropriate branch for $z^{1/n}$ and then raises this result to the mth power.

The **Transfer Save State** command saves the current branch selection as part of the state initialization file it creates (see Section 2.14).

4.6 Factoring Expressions

Use the **Factor** command or the FACTOR function to factor an expression or subexpression with respect to some or all of its variables. This command prompts you to select an expression or subexpression to factor. If you highlight a subexpression, the result is a copy of the whole expression in which only the subexpression is factored.

If an expression is a number, the **Factor** command factors the number into a product of **prime numbers** raised to integer powers. For example,

1234567890

factors to

$$2 \cdot 3^2 \cdot 5 \cdot 3607 \cdot 3803$$

Factoring a number may take a long time if it has two or more prime factors, each with more than six decimal digits. Press **Esc** to abort a **Factor** command that is taking too long.

The general objective when factoring an expression with respect to designated variables is to maximize the number of independent factors in which the designated variables occur linearly and which have the simplest possible coefficients. There is often a tradeoff between these two objectives. For example, linear factors might be achievable only at the expense of complex coefficients.

A univariate expression is factored with respect to its variable. Its numeric coefficients and exponents are not factored. To factor them, highlight and factor them individually *after* factoring the entire expression with respect to the variable.

The **Factor** command displays the selection field

FACTOR: Amount: Trivial Squarefree **Rational** raDical Complex

and waits for you to select the amount of factoring desired. Successive options request increasing amounts of factorization at the expense of increased computation time.

The FACTOR function can be used on the author line to factor an expression when it is simplified. The call FACTOR(*u*, *amount*, *x*, *y*, ...) factors expression *u*(*x*, *y*, ...) with respect to the variables *x*, *y*, ... If no variables are given, expansion is done with respect to all the variables in the expression. The amount argument (Trivial, Squarefree, Rational, raDical, or Complex) specifies the amount of factoring done on *u*. *amount* defaults to Rational.

Trivial factoring places expressions over a **common denominator**. For example, if Trivial is selected, then

$$3 - \frac{1}{x + 2}$$

factors to

$$\frac{3 \cdot x + 5}{x + 2}$$

Trivial factoring also factors out numbers and least powers of variables. For example, if Trivial is selected, then

$$2 \cdot x^3 - 12 \cdot x^2 + 18 \cdot x$$

factors to

$$2 \cdot x \cdot (x^2 - 6 \cdot x + 9)$$

Square-free factoring does trivial factoring and factors powers of sums or products of *different* powers of sums. For example, if Squarefree is selected, the previous example factors to

$$2 \cdot x \cdot (x - 3)^2$$

As another example, if Squarefree is selected,

$$x^4 + 2 \cdot x^3 - 3 \cdot x^2 - 8 \cdot x - 4$$

factors to

$$(x + 1)^2 \cdot (x^2 - 4)$$

Note that x^2–4 was not factored since x+2 and x–2 are the *same* power (i.e. the first power) of sums.

Rational factoring does square-free factoring and factors products of sums without introducing new fractional powers or complex numbers. For example, if Rational is selected, the previous example factors to

$$(x + 2) \cdot (x - 2) \cdot (x + 1)^2$$

As another example, if Rational is selected,

$$4 \cdot x^2 - 9$$

factors to

$$(2 \cdot x + 3) \cdot (2 \cdot x - 3)$$

Radical factoring does rational factoring with the additional possibility of introducing fractional powers of numbers, such as $\sqrt{2}$. For example, if raDical is selected

$$x^2 - 2$$

factors to

$$(x + \sqrt{2}) \cdot (x - \sqrt{2})$$

in exact mode, or to

$$(x + 1.41421) \cdot (x - 1.41421)$$

in approximate or mixed mode.

Complex factoring does radical factoring with the additional possibility of introducing new complex numbers. For example, if Complex is selected

$$x^2 + 2$$

factors to

$$(x + \sqrt{2} \cdot i) \ (x - \sqrt{2} \cdot i)$$

in exact mode, or to

$$(x + 1.41421 \cdot i) \ (x - 1.41421 \cdot i)$$

in approximate or mixed mode.

Fractional powers of complex numbers may introduce trigonometric and inverse trigonometric functions to achieve normal rectangular form (see Section 4.5). In approximate or mixed mode, such subexpressions simplify to approximate real or complex numbers for numeric arguments. In exact mode, they are left in symbolic form, which may be quite bulky.

Use radical factoring if you want real factors that are factored as far as possible. Use complex factoring if you want factors that are linear in a variable. Do *not* use radical or complex factoring if you want a concise expression without new fractional powers and/or complex numbers.

Exercise: Factor the expression

$$3 \cdot x^7 + 18 \cdot x^6 + 48 \cdot x^5 + 126 \cdot x^4 + 135 \cdot x^3 - 324 \cdot x^2 - 486 \cdot x$$

using each of the five levels of factoring. Compare the relative compactness and comprehensibility of the five factored expressions with each other and with the original unfactored expression.

A multivariate expression can be factored with respect to some or all of its variables. The **Factor** command displays a list of the variables on the message line, and you are prompted to enter the desired **factorization variables**. Press **Enter** if you want factorization with respect to all of the variables in the displayed order. Otherwise enter the desired primary factorization variable, the secondary factorization variable, etc. These factorization variables temporarily override the ordering specified by the variable order list (see Section 4.3).

Subexpressions that do not contain factorization variables are simplified without being unnecessarily transformed. In other words, subexpressions involving only non-factoring variables are neither expanded nor factored unnecessarily. For example, consider the polynomial

$$x^2 \cdot y^2 - x^2 - y^4 + y^2$$

- Rational factoring with respect to x as the primary variable and y as the secondary variable yields

$$(x + y) \cdot (x - y) \cdot (y + 1) \cdot (y - 1)$$

- Rational factoring with respect to x alone yields

$$(x + y) \cdot (x - y) \cdot (y^2 - 1)$$

Similar powers of factoring variables are collected and simplified. For example, rational factoring of

$$x^2 \cdot y^4 - 2 \cdot x^2 \cdot y^2 + x^2 + 2 \cdot x \cdot y^2 - 2 \cdot x + 1$$

with respect to x alone yields

$$(x \cdot (y^2 - 1) + 1)^2$$

New radicals introduced by radical or complex factoring may include non-factorization variables as well as numbers. For example, radical factoring

$$a \cdot x^2 + b \cdot x + c$$

with respect to x alone yields

$$a \cdot \left[x + \frac{\sqrt{(b^2 - 4 \cdot a \cdot c)} + b}{2 \cdot a}\right] \cdot \left[x - \frac{\sqrt{(b^2 - 4 \cdot a \cdot c)} - b}{2 \cdot a}\right]$$

Depending on the expression, the type of factoring, and the factorization variables, it may take a long time to factor an expression. Press **Esc** to abort a **Factor** command that is taking too long. You can then try factoring only portions of the expression or factoring with respect to a different variable. These actions may drastically reduce the computing time and improve the comprehensibility of a result.

If the **Factor** command generates a large expression that is not of interest, use the **Remove** command (see Section 3.7) to save computer memory and screen space.

Note that both the **Factor** (when not introducing fractional powers) and **Simplify** commands transform an expression into a sufficiently simple form (see Section 4.2). However, the **Simplify** command is often much faster and usually yields an expression more similar to the original expression.

Exercise: Enter and rationally factor x^n-1 for $n = 2, 3, 4, \ldots$ until you can predict the form of the answer and the approximate computing time. Repeat the exercise for x^n+1.

Exercise: Expand and then rationally factor $(x+1)^n$ for $n = 2, 3, 4, \ldots$ until you can predict the approximate time required to factor the expanded form of the expression.

4.7 Partial Fraction Expansion

Use the **Expand** command to expand an expression or subexpression with respect to some or all of its variables. If a polynomial expression is expanded, the polynomial expansion is returned (see Section 4.4). If a rational expression with a denominator containing any of the expansion variables is expanded, the partial fraction expansion is returned.

The EXPAND function can be used on the author line to do partial fraction expansion. The call EXPAND(*u*, *amount*, *x*, *y*, ...) does partial fraction expansion on *u*(*x*, *y*, ...) with respect to the variables *x*, *y*, ... If no variables are given, expansion is done with respect to all the variables in the expression. The amount argument (Trivial, Squarefree, Rational, or raDical) specifies the amount of factoring done on the denominator of *u*. *amount* defaults to Rational.

When a rational expression is expanded, the following sequence of transformations occurs:

- If the rational expression is improper with respect to the primary expansion variable, it is transformed using polynomial division into an expanded polynomial quotient plus a **proper rational** expression. (A rational expression is proper with respect to a variable if the *total* degree of the variable in the numerator of the expression is less than its *total* degree in the denominator);
- The denominator of the proper expression is factored (see Section 4.6) with respect to the expansion variables. The amount of factoring is that specified by the **Expand** command or by the *amount* argument of the EXPAND function;
- The proper expression is expanded into a sum of independent **super-proper rational** terms. The denominator of each of these terms is one of the factors of the proper expression's denominator raised to a power not exceeding its power in the denominator. (A rational expression is super-proper with respect to a variable if the *total* degree of the variable in the numerator of the expression is less than its degree in the *base* of the denominator);
- If the numerator of a super-proper rational term is a sum that contains expansion variables, the denominator of that super-proper rational term is distributed over those terms of the numerator that contain expansion variables.

For example, expanding

$$\frac{25 \cdot x^4 + 81 \cdot a \cdot x^2 + 324 \cdot a \cdot x + 324 \cdot a}{x^3 + x^2 - 8 \cdot x - 12}$$

with respect to x yields

$$\frac{81\cdot(a+1)}{x-3} - \frac{80}{(x+2)^2} + \frac{144}{x+2} + 25\cdot x - 25$$

Note that the degree of x in the numerator of the first term is 0, which is not only less than the total degree of x in the denominator but less than the degree of x in $x+2$, the base of the denominator. Thus the term is super-proper as required. Also note that $1/(x-3)$ was *not* distributed over $81(a+1)$ since $81(a+1)$ does not contain an expansion variable.

Partial fraction expansion can be used to determine the quotient and remainder of two polynomials with respect to the primary expansion variable as follows:

- The quotient polynomial is the sum of those terms of the partial fraction expansion that do *not* contain the primary variable in their denominators. If you wish to collect the quotient terms, use a **Build** command or an **Author** command in conjunction with the **F3** function key.
- The remainder polynomial is the difference of the numerator polynomial and the product of the quotient and denominator polynomials. Thus if you also want the remainder polynomial, use a **Build** command to construct and simplify this difference.

The rational factoring required of the denominator can make partial fraction expansion quite slow if the denominator has sums of degree exceeding 2 in any of the expansion variables. Consequently, expand only with respect to variables that you must (usually only a single variable).

If you want a ratio of an expanded numerator over an expanded denominator rather than partial fractions, then separately highlight and expand the numerator and denominator. This kind of expansion is usually much quicker than partial fractions — at least for ratios with a sum in the denominator having expansion variables raised to high degrees. However, the resulting expression is rarely informative or useful: It ordinarily exhibits neither the zeros, poles, nor residues of the ratio, and it does not partition the ratio either into many factors or many terms.

4.8 Substituting Values

Use the **Manage Substitute** command to substitute replacement values for all occurrences of the variables in an expression or for all occurrences of one subexpression in an expression. This command prompts you to select an expression or subexpression.

If you select an *entire* expression, the **Manage Substitute** command displays a message of the form

Enter replacement for x

and waits for you to enter a replacement value for each variable x in the expression. Initially the name of the variable is displayed on the entry line. Thus for variables that are *not* to be replaced, just press **Enter**. For variables that are to be replaced, type the replacement value over the variable names. Press **F3** to enter the highlighted expression or subexpression on the entry line.

Exercise: Use the **Manage Substitute** command to substitute 2 for a, 3 for b, and 5 for c in the expression

$$a \cdot x^2 + b \cdot x + c$$

Exercise: Use the **Manage Substitute** command to substitute $x+1$ for x in the expression used in the previous exercise. Then expand the result.

If you select (highlight) a *sub*expression, the **Manage Substitute** command displays the message

Enter replacement for subexpression

and waits for you to enter a replacement value for the subexpression. Type the replacement value or press **F3** to enter the highlighted expression or subexpression on the entry line. Every instance of the originally highlighted subexpression is replaced by the new value. This is useful for making an expression more compact by substituting variables for subexpressions that are large and/or occur repeatedly in the expression.

Exercise: Use the **Manage Substitute** command to substitute r for the subexpression

$$\sqrt{(x^2 + y^2)}$$

in the expression

$$\frac{3\cdot\sqrt{(x^2+y^2)}}{2+\sqrt{(x^2+y^2)}}$$

The result of a **Manage Substitute** command is not automatically simplified, so you can see the unsimplified result before issuing a **Simplify** command. To combine simplification with substitution, press **Ctrl-Enter** or **Ctrl-J** instead of **Enter** after typing the last substitution value.

When substituting for variables in an expression, the substitutions are done simultaneously for every variable in the expression. This makes substitution a convenient way to interchange the variables in an expression. For example, substituting y for x and x for y in

$$x + 2\cdot y$$

yields

$$y + 2\cdot x$$

Exercise: Use the **Manage Substitute** command to interchange a and c within the expression

$$a\cdot x^2 + b\cdot x + c$$

When substituting for a subexpression in an expression, a substitution is made only for exact matches of the subexpression within the expression. For example, substituting k for the subexpression t^3 in the expression

$$t^6 + t^3$$

yields

$$t^6 + k$$

rather than

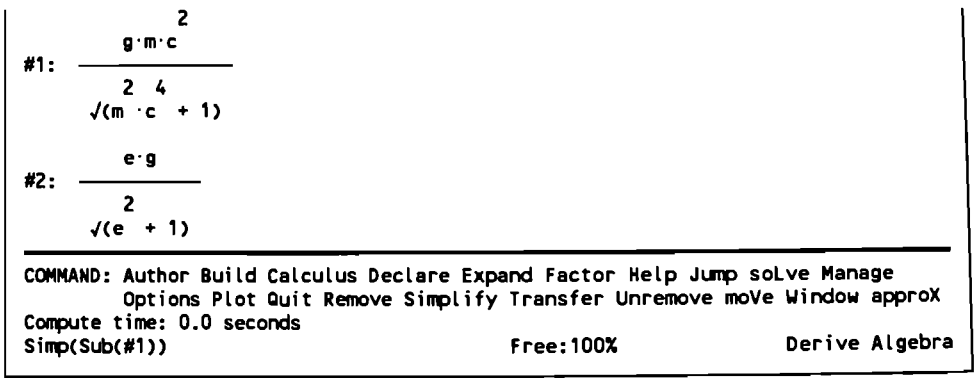
$$k^2 + k$$

To obtain the latter expression, substitute $k^{1/3}$ for t in the original expression and simplify. Similar remarks apply when substituting for product and sum subexpressions. In general, to algebraically substitute the expression R for every instance of subexpression S in expression E, solve the equation $S = R$ for some variable in S. Then substitute the solution for that variable in expression E.

Exercise: Use the **Manage Substitute** command to algebraically substitute e for mc^2 in the expression

$$\frac{g \cdot m \cdot c^2}{\sqrt{(m^2 \cdot c^4 + 1)}}$$

The screen should look similar to:


#1: g·m·c^2 / √(m^2·c^4 + 1)
#2: e·g / √(e^2 + 1)
COMMAND: Author Build Calculus Declare Expand Factor Help Jump soLve Manage
Options Plot Quit Remove Simplify Transfer Unremove moVe Window approX
Compute time: 0.0 seconds
Simp(Sub(#1))
Free:100%
Derive Algebra

Exercise: Use the **Manage Substitute** command to algebraically substitute z for $x+y$ in the expression

$$\frac{x + y + 3}{x + y + 5}$$

Provided there are no fractional powers, substitution into factored expressions is usually faster than substitution into expanded expressions. In approximate mode, substitution into such factored expressions is usually also more accurate than substitution into expanded expressions.

Substitution does not fully commute with simplification. For example, substituting -1 for x in the *un*simplified ratio

$$\frac{x^2 + 2 \cdot x + 1}{x^2 - 1}$$

yields

$$\frac{(-1)^2 + 2\cdot(-1) + 1}{(-1)^2 - 1}$$

which simplifies to the indeterminate form 0/0, which displays as a question mark (see Section 4.9).

In contrast, first simplifying the original ratio cancels the polynomial greatest common divisor $x+1$ giving the simplified ratio

$$\frac{x + 1}{x - 1}$$

Then substituting -1 for x in this simplified ratio yields

$$\frac{-1 + 1}{-1 - 1}$$

which simplifies to 0. Note that 0 is the limit of the original ratio as x approaches -1, making this a **removable singularity**.

Cancellation of a polynomial greatest common divisor replaces the removable singularities at the zeros of the greatest common divisor with their limits. So, substitution after simplification may yield a result that simplifies to a unique value, whereas simplification after substitution yields an indeterminate form.

Usually it is desirable to remove removable singularities from an expression. This is done by defining the value of the expression to be its limit at these singularities. Thus if substitution into an unsimplified expression yields an indeterminate form, try simplifying the expression *before* making the substitution.

4.9 Entering Infinite Values

DERIVE knows how to handle infinite values if they should arise during expression simplification. Infinite values can entered on the author line or generated during simplification. On PC compatible computers, infinity displays as ∞ and can be entered by pressing **Alt-0**. (Note that infinity is the **Alt**ernative to **0**.)

In addition to its magnitude, the sign or, more generally, the phase angle of an infinite value must be specified. The following are the infinite values recognized by ***DERIVE***:

- Plus infinity is entered as inf and displays as ∞.
- Minus infinity is entered as –inf and displays as $-\infty$.
- Plus-or-minus infinity is entered as "+-" inf and displays as $\pm\infty$.
- Imaginary infinity is entered as #i inf and displays as $\hat{\imath}\cdot\infty$.
- Complex infinity is entered as unit_circle inf and displays as unit_circle·∞.

The $\pm$ (plus-or-minus) operator indicates that an expression has a specific magnitude but its sign is unknown. For example, TAN(π/2) and 1/0 simplify to $\pm\infty$ instead of ∞ since their sign can be either positive or negative. Similarly SIGN(0) simplifies to ± 1 instead of 1 since the sign of 0 depends on how it is approached.

The $\pm$ operator can be entered on the author line by typing "+–" (the double quote marks must be included), or by pressing **Alt** – (minus) on PC compatible computers.

If simplification of an expression results in a value of unknown phase and possibly finite or infinite magnitude, it is displayed as a question mark. For example, $\infty-\infty$, ∞/∞, and 0/0, simplify to ?

Exercise: Enter and simplify various expressions involving infinity such as -3∞, $\infty+5$, $(-\infty)^3$, and $(1/0)^{-1}$.

If an expression simplifies to ? after you substitute a value for a variable, you can try using a **Calculus Limit** command to see if the expression approaches a limit as the variable approaches the value (see Section 7.1).

Exercise: Enter SIN(x)/x, use a **Manage Substitute** command to replace x with 0 in the expression, and then simplify the result. Then find the limit of the original expression as x approaches 0 using a **Calculus Limit** command.

4.10 Declaring Domains of Variables

Many transformations are valid for both real and complex expressions. For example, the transformation

$$(\sqrt{z})^2 \rightarrow z$$

is valid whether z is a real expression or a complex expression. However, some transformations are valid only for real expressions. For example, the transformation

$$\sqrt{(x^2)} \rightarrow |x|$$

is valid *only* if x is real.

Some transformations are valid only for a restricted interval of real values. For example,

$$\sqrt{(x \cdot y)} \rightarrow \sqrt{x} \cdot \sqrt{y}$$

is valid for real x and y unless both are negative.

Exercise: Verify that the above transformation is valid if x is 1 and y is -1, and that it is invalid if both x and y are -1.

Some transformations are only valid for integer-valued expressions. For example, SIN($n\pi$) can simplify to 0 only if n has an integer value.

Even as basic a transformation as commuting the operands of a product is valid only if the operands are scalars.

When simplifying an expression, ***DERIVE*** exploits a transformation only if it can determine that the transformation is valid. This determination depends heavily on the declared domain of the variables in the expression.

Since real-valued variables are the most common, the default domain of variables is real. However, if an expression contains some variables that only assume integer values or a subset of real values, you can restrict the domain of these variables. On the other hand, if an expression contains some variables that can assume complex values or nonscalar values, you can expand the domain of these variables.

Use the **Declare Variable** command to declare the default domain of variables or to declare the domain of a particular variable. This command is also used to assign variable values (see Section 4.11). Use the **Transfer Clear Variables** command to clear all variable domain and value declarations (see Section 2.9).

Declared variable names can be multi-character even in character input mode (see Section 4.1). The **Declare Variable** command can turn the name of a user-defined function back into a variable name. However, it cannot turn the name of a pre-defined function or constant (see Chapter 6) into a variable name.

The **Declare Variable** command displays the data entry field

DECLARE VARIABLE name:

After you enter a variable name, the selection field

DECLARE VARIABLE: Value Integer **Real** Complex Nonscalar

is displayed and ***DERIVE*** waits for you to select Value (see Section 4.11) or one of the following domains for the variable:

- Integer - an integer-valued variable;
- Real - a real-valued variable;
- Complex - a complex-valued variable; or
- Nonscalar - a nonscalar-valued variable (i.e. a vector or matrix).

If you select Integer or Real, the selection field

DECLARE VARIABLE: **All** Positive Negative nonpoSitive nonneGative Interval

is displayed and ***DERIVE*** waits for you to select one of the following intervals for the variable:

- All - a variable between minus infinity and plus infinity;
- Positive - a variable greater than zero;
- Negative - a variable less than zero;
- nonpoSitive - a variable less than or equal to zero;
- nonneGative - a variable greater than or equal to zero; or
- Interval - a variable with a user-specified domain.

If you select Interval, the current domain of the variable is displayed as an inequality of the form

$$lower\ (<)\ \leq\ variable\ (<)\ \leq\ upper$$

where *variable* is the variable name you entered, and *lower* and *upper* are the variable's current bounds. Parentheses around < indicate a **strict bound** (the variable

can never equal the bound). Parentheses around ≤ indicate a **nonstrict bound** (the variable can equal the bound). Using the **Tab** key to move between fields, you can enter numbers, or plus or minus infinity (see Section 4.9) for the bounds and/or select the strictness of the bounds using the **Space** bar. If you press **Esc**, the command is aborted.

When a domain declaration is made to the variable x and no interval is specified, the **Declare Variable** command generates an expression of the form

$$x :\epsilon \textit{ domain}$$

where *domain* is Integer, Real, Complex, or Nonscalar. If the domain is integer or real and an interval is specified, the command generates an expression of the form

$$x :\epsilon \textit{ domain interval}$$

where *interval* is an interval expressed in standard notation. Parentheses ( ) denote open intervals, square brackets [ ] denote closed intervals, and a combination denotes half-open intervals.

Domain declarations can also be made on the author line by entering expressions of the above form using upper and lower case exactly as shown. The epsilon character can be entered by spelling it out in full or by pressing **Alt-N** on PC compatible computers (see Section 4.1). For example, entering

```
p :ϵ Real [0, ∞)
```

declares p to be a nonnegative real-valued variable.

The transformation

```
x - x  →  0·x  →  0
```

is essential for simplifying expressions. Since this and other transformations are questionable if x is infinity, ***DERIVE*** does not permit variable domains to include an infinite magnitude. Thus infinite bounds are always strict.

Although variables cannot be declared infinite, you can substitute an infinite value for a variable in an expression (see Sections 4.8 and 4.9). However, you risk an invalid result if the expression was derived using transformations, such as the above, which are questionable for infinite values.

As might be expected, ***DERIVE*** simplifies both x/x and x^0 to 1, even if the domain of the variable x includes 0. For consistency, ***DERIVE*** also simplifies 0^0 to 1, since this is just a special case of the above x^0 simplification.

The domains of variables are used during simplification of fractional powers, absolute values, logarithms, and some other irrational functions. They do not restrict the values that can be substituted for variables (see Section 4.8). Thus, you can substitute a complex value for a real variable or a negative value for a positive variable. Such substitutions are made at your own risk.

Normally variables are assigned values using the **Declare Variable** command and selecting the Value option (see Section 4.11). An alternative method is using a degenerate domain declaration. For example, the domain declaration

$$7 \le y \le 7$$

assigns *y* the value 7.

Exercise: Enter and simplify

$$\sqrt{(z^2 - 2 \cdot z + 1)}$$

with *z* declared real, with *z* declared ≥1, and with *z* declared complex. Note how restricted domains can give simpler results.

4.11 Assigning Values to Variables

Use the **Declare Variable** command to declare a variable and optionally assign it a value or clear an existing value. Previous value assignments and domain declarations are over-ridden. This command is also used to declare variable domains (see Section 4.10). Use the **Transfer Clear Variables** command to clear all variable domain and value declarations (see Section 2.9).

Declared variable names can be multi-character even in character input mode (see Section 4.1). The **Declare Variable** command can turn the name of a user-defined function back into a variable name. However, it cannot turn the name of a pre-defined function or constant (see Chapter 6) into a variable name.

The **Declare Variable** command displays the data entry field

DECLARE VARIABLE name:

After you enter a variable name, the selection field

DECLARE VARIABLE: Value Integer **Real** Complex Nonscalar

is displayed. If you select the Value option, the data entry field

DECLARE VARIABLE value:

is displayed and ***DERIVE*** waits for you to enter an expression. If you leave the line blank and just press **Enter**, the name becomes an unassigned variable. If you enter an expression, the name becomes an assigned variable having the expression as its value. If you press **Esc**, the command is aborted.

While entering the expression, you can use all the normal line editing features provided by the **Author** command. In particular, you can use the **F3** and **F4** function keys to insert existing expressions and subexpressions on the author line.

Instead of a **Declare Variable Value** command, you can use an **Author** command to declare a variable and optionally assign it a value or clear an existing value. On the author line, enter: the variable name; the : = **assignment operator**; and the variable's value, if it is to be an assigned variable.

For example, entering

area_circle := $\pi \cdot r^2$

assigns the variable *area_circle* the value $\pi \cdot r^2$.

When an expression containing an assigned variable is simplified, all occurrences of the variable are replaced by its value. For example, if *area_circle* is assigned as above,

$$2 \cdot \text{area\_circle} + 2 \cdot \pi \cdot h \cdot r$$

simplifies to

$$2 \cdot \pi \cdot h \cdot r + 2 \cdot \pi \cdot r^2$$

This expression gives the surface area of a cylinder of height h and radius r. In turn, it can be assigned as the value of the variable *area_cylinder*.

Entering

$$\text{area\_circle} :=$$

clears any value previously assigned to *area_circle* and makes it an unassigned variable.

Expression label numbers can be used like variable names on the author line or when ***DERIVE*** commands prompt for an expression (see Section 3.3). Also previous expressions and subexpressions can be referenced by highlighting. For these reasons variable assignments are usually unnecessary when using ***DERIVE***.

On the other hand, variable assignments are convenient to reference an expression after it scrolls off the screen or to assign a descriptive name to an expression that is of special significance.

Variable assignments that you forget about can lead to surprising or incorrect results. Thus using multi-character names for assigned variables is recommended since it reduces the chance of name conflicts with unassigned variables.

Exercise: Use the **Declare Variable Value** command to assign *radius* the value s^2, and then assign s the value 5. Then enter and simplify *radius*. Note that 25 rather than s^2 is returned. This shows that ***DERIVE*** repeatedly applies variable assignments until a constant or unassigned variable is reached.

Exercise: Use the := assignment operator to assign the variable *pn* the value $pn+1$. Next, enter and simplify *pn*. Then keep simplifying the result until you see how ***DERIVE*** avoids the infinite recursion that would occur if ***DERIVE*** did not check for it.

DERIVE includes two **units conversion** files and a **physical constants** file that provide good examples of the use of assigned variables:

- ENGLISH.MTH assigns values to numerous English units so that expressions involving these units simplify to ones expressed in terms of feet, pounds, and seconds.
- METRIC.MTH assigns values to numerous Metric units so that expressions involving these units simplify to ones expressed in terms of meters, grams, and seconds.
- PHYSICAL.MTH assigns highly accurate values to numerous fundamental physical constants.

Use the **Transfer Merge** command (see Section 2.9) to load these files and to see exactly which units and physical constants are defined. Loading both units conversion files converts quantities to the standard units of the *last* file that was loaded. For example, if just ENGLISH.MTH is loaded,

$$5 \cdot \frac{\text{furlong}}{\text{fortnight}}$$

simplifies to

$$\frac{11 \cdot \text{foot}}{4032 \cdot \text{second}}$$

and approximates to

$$\frac{0.00272817 \cdot \text{foot}}{\text{second}}$$

If METRIC.MTH is now loaded, either of the above expressions approximates to

$$\frac{8.31547 \cdot 10^{-4} \cdot \text{meter}}{\text{second}}$$

After loading these files, you can use the **Remove** command (see Section 3.7) to remove the expressions from the algebra window but still use the variable assignments made in the files. Use the **Transfer Clear Variables** command to totally clear the variable assignments thereby freeing more memory (see Section 2.9).

4.12 Defining Functions

While using *DERIVE*, you may find it necessary to perform a sequence of operations repeatedly or frequently. As a shortcut, you can define functions in terms of the pre-defined functions and operators. Use the **Transfer Save** command to save these function definitions in an MTH file (see Section 2.8). Such files eliminate the time required to redeliver and reenter formulas. Some of the MTH files distributed with *DERIVE* are just such collections of function definitions.

Use the **Declare Function** command to declare a function and optionally assign it a definition, or to clear an existing definition. This command can turn a variable name into a function name. *DERIVE*'s pre-defined functions and constants (see Chapter 6) can *not* be redefined.

You can use the **Declare Variable Value** command to clear a user-defined function and make it a variable name (see Section 4.11). The **Transfer Clear Functions** command clears all user-defined functions (see Section 2.9).

In case-insensitive mode (see Section 4.1), both user and pre-defined function names are displayed in upper-case letters. Variable names are displayed in lower-case letters.

The **Declare Function** command displays the data entry field

DECLARE FUNCTION name:

Function names can be multi-character even in character input mode (see Section 4.1). After you enter a function name, the data entry field

DECLARE FUNCTION definition:

is displayed and *DERIVE* waits for you to enter an expression. If you leave the line blank and just press **Enter**, the name becomes an arbitrary function, and you are prompted to enter its variables. If you enter an expression, the name becomes a user-defined function having the expression as its definition. If you press **Esc**, the command is aborted.

While entering an expression, you can use all the normal line editing features provided by the **Author** command. In particular, you can use the **F3** and **F4** function keys to insert existing expressions and subexpressions on the author line.

Instead of a **Declare Function** command, you can use an **Author** command to declare a function and optionally assign it a definition or clear an existing definition. On the author line, enter: the function name and a list of some or all of the variables in the function's definition, enclosed in parentheses and separated by commas; then enter the **:= assignment operator** and the function's definition. Omit the definition for arbitrary functions.

For example, entering

```
ACCELERATION(f, m) := f/m
```

assigns the function ACCELERATION(*f*, *m*) the definition *f*/*m*. Note that unlike the **Declare Function** command, entering a function definition using an **Author** command allows you to specify the order and number of the function's arguments.

When an expression containing calls to a function is simplified, all function calls are replaced by a copy of its definition in which the variables have been replaced by the values specified in the call on the function. For example, if ACCELERATION is defined as above, then

```
ACCELERATION(6, 2)
```

simplifies to 3. If a function is called with fewer arguments than there are variables in its definition, the excess variables are not replaced. For example,

```
ACCELERATION(6)
```

simplifies to 6/*m*.

Entering

```
ACCELERATION(f, m) :=
```

clears any definition previously assigned to ACCELERATION and makes it an arbitrary function.

Exercise: Use the **Transfer Clear All** command to clear the algebra window. Then use the **Declare Function** command to define the function HYPOTENUSE(*a*, *b*) that returns the length of the hypotenuse of a right triangle, where the length of the other sides are *a* and *b*. Next, using your new function, find the hypothenuse of the triangle whose other sides are 3 and 4. Finally, use a **Transfer Save** command to save the function definition in a file named HYPOT.MTH (see Section 2.8).

> **Hint**: The square of the hypotenuse is equal to the sum of the squares of the other two sides.

4.13 Entering Equations and Relations

Equations and inequalities are in a class of expressions called relations. In *DERIVE*, a relation consists of two expressions separated by one of the following relational operators: = (equal), /= (not equal), < (less than), <= (less than or equal), > (greater than), and >= (greater than or equal). On PC compatible computers, <= is displayed as ≤, and >= is displayed as ≥.

To enter a relation using an **Author** command, enter the left side of the relation, the relational operator, and then the right side. You can also build equations using the = operator provided by the **Build** command (see Section 3.5).

The **Simplify** command independently simplifies the two sides of a relation. For example, the equation

$$x + 2 \cdot x = c + x \cdot x$$

simplifies to

$$3 \cdot x = x^2 + c$$

The remainder of this section illustrates how the **Simplify**, **Build**, and **Factor** commands can be used to solve relations manually by issuing a sequence of commands. This procedure is useful for learning how to solve relations. Skip to the next section if you want to automate the process of solving them.

If an equation is added to an equation, the result is an equation whose left side is the sum of the left sides of the original equations and whose right side is the sum of the right sides of the original equations. Similar remarks apply to subtracting, multiplying, and dividing an equation by an equation.

For example, to manually solve the linear equation

$$2 \cdot x - 3 = 5$$

for x, enter it and the equation

$$3 = 3$$

Then use the **Build** command to add the two equations and simultaneously simplify the result by pressing **Ctrl-Enter** or **Ctrl-J**, giving

$$2 \cdot x = 8$$

Next, enter the equation

$$2 = 2$$

Finally, use the **Build** command to divide *and* simplify the two equations, giving

$$x = 4$$

If an expression (i.e. a non-equation) is added to an equation, the expression is added to both sides of the equation. Similar remarks apply to subtracting, multiplying, and dividing an equation by an expression. The ability to combine equations with non-equations makes it easier to manually solve equations.

For example, to more easily solve the linear equation

$$2 \cdot x - 3 = 5$$

for *x*, use the **Build** command as follows:

- issue the **Build** command, highlight the whole equation, and press **Enter**,
- select the + operator by pressing the + key,
- highlight the 3 in the equation and press **Enter**,
- select the / operator by pressing the / key,
- highlight the 2 in the equation and press **Enter**, and
- select Done by pressing **Ctrl-Enter** or **Ctrl-J**.

Quadratic equations can also be solved manually by completing the square. For example, to solve the quadratic equation

$$x^2 + 5 \cdot x + 6 = 0$$

for *x*, enter it and subtract 6 from both sides by authoring and simplifying the expression *#n*–6, where *n* is the label number of the equation, giving

$$x^2 + 5 \cdot x = -6$$

Then complete the square by adding $(5/2)^2$ to both sides of the equation by authoring and factoring the expression *#n*+(5/2)^2, where *n* is the label number of the above equation, giving

$$\frac{(2 \cdot x + 5)^2}{4} = \frac{1}{4}$$

Then multiply both sides of the equation by 4 by authoring and simplifying the expression $4 \cdot \#n$, where n is the label number of the above equation, giving

$$(2 \cdot x + 5)^2 = 1$$

Taking the square root of both sides of this equation gives the two linear equations

$$2 \cdot x + 5 = 1$$

$$2 \cdot x + 5 = -1$$

that can be solved as described previously.

Quadratic and higher order polynomial equations can be solved by collecting all the terms on one side of the equation and factoring. For example, rationally factoring the equation

$$x^2 - 5 \cdot x + 6 = 0$$

yields the equation

$$(x - 2) \cdot (x - 3) = 0$$

Then set the factors equal to 0 by highlighting each factor and using the **Build** command. This gives the two linear equations

$$x - 2 = 0$$

$$x - 3 = 0$$

that can be solved as described previously.

Exercise: Use the **Factor** and **Build** commands to manually solve

$$3 \cdot y \cdot z^2 - 2 \cdot y \cdot z = 2 \cdot x - 3 \cdot x \cdot z$$

for z.

The result of applying a function to an equation is an equation whose left and right sides are the result of independently applying the function to the left and right sides of the original equation. For example, to solve the equation

$$\mathrm{LN}(x) = 5$$

where LN is the natural logarithm function (see Chapter 6), use the **Build** command to apply the exponential or inverse logarithm function EXP to the equation. This gives

$$\mathrm{EXP}(\mathrm{LN}(x) = 5)$$

which simplifies to

$$x = e^5$$

Be aware that applying an inverse trigonometric function (see Section 6.4) to an equation discards all solutions not on the principal branch. Operations such as clearing the denominator or squaring both sides of an equation can introduce **spurious solutions**. For example, squaring both sides of the equation

$$\sqrt{x} = -1$$

yields $x=1$, which is not a solution. Solutions should always be verified by substituting them into the original relation.

Relations are *not* automatically regarded as simultaneous facts, and they are *not* automatically tested for truth or consistency. Think of a ***DERIVE*** algebra window as a blackboard containing relatively independent expressions, including relations. Some of these expressions may be derived from earlier ones. However, others may have no relation to each other. Variables may mean different things in different parts of a problem or a set of problems. You can choose to combine subsets of these expressions using the **Build** command, while ignoring other perhaps contradictory expressions.

4.14 Solving Equations and Relations

Use the **soLve** command to solve a relation for a particular variable or to find the zeros of an expression. Type the letter L to issue this command since there is another menu option starting with the letter S.

The **soLve** command prompts you to select an expression. If you select an expression that is *not* a relation, the **soLve** command returns the zeros of the expression (i.e. the values of the variable that make the expression equal to 0). If the expression you select has more than one variable, the **soLve** command prompts you to enter the variable that you want to solve for in terms of the other variables.

Instead of a **soLve** *command*, you can use the SOLVE *function* to solve a relation for a variable. On the author line enter SOLVE, a left parenthesis, the relation, the variable, and a right parenthesis. Unlike the command, the SOLVE function returns a vector of solutions. For example,

$$\text{SOLVE}\ (-2\cdot x + 3\cdot y \le 7,\ x)$$

simplifies to

$$\left[x \ge \frac{3\cdot y - 7}{2}\right]$$

If precision is in exact mode, the **soLve** command attempts to solve the relation for *all* of its real and complex solutions. The set of solutions returned is *not* restricted by the declared domains of the solution variables (see Section 4.10). The **soLve** command can directly solve binomial, quadratic, cubic, and quartic equations. It can also solve higher degree polynomial equations that can be exactly factored into such equations.

Each solution the **soLve** command determines is displayed as a relation with the variable on the left and an expression free of that variable on the right. For example, solving the equation

$$x^2 - 5\cdot x + 6 = 0$$

for x results in the equations $x = 2$ and $x = 3$.

To help solve equations involving trigonometric functions, try various settings of the **Manage Trigonometry** command (see Section 6.3).

If the given equation or inequality is too difficult to solve completely, one or more implicit relations are displayed as an equation or an inequality with an expression containing the variable in an inextricable way on the left and a zero on the right. For example, solving the equation

$$3^x = x^2$$

for x results in the equation

$$3^x - x^2 = 0$$

Exercise: Use the **soLve** command to solve the inequality

$$|3 \cdot x + 2| > 5$$

for x.

If precision is in approximate mode *and* an equation contains only one variable, the **soLve** command attempts to numerically compute an approximate *real* solution to the equation. (Functions are defined in SOLVE.MTH for computing an approximate *complex* solution to an equation (see Section 9.1)).

In approximate mode, the **soLve** command prompts you to enter the lower and upper bounds of an interval to search for a solution. If you have no idea where a solution occurs, plot the difference in the two sides of the equation to find an interval where the plot crosses or touches the x-axis (see Section 5.1).

To find a solution other than ones already found, specify an interval that excludes the known solutions.

Exercise: In approximate mode, use plotting and the **soLve** command to find the first three positive solutions to the equation

$$x \cdot \mathrm{SIN}(x) = 1$$

The **soLve** command will find an approximate solution if the difference in the two sides of the equations is continuous and has different signs at the ends of the interval. Even if the equation is discontinuous in the interval or does not have different signs at the ends, the **soLve** command usually finds a solution if one exists in the interval.

DERIVE is normally able to return a solution accurate to the current digits of precision specified by the **Options Precision** command (see Section 3.8). However, cancellation of approximate large magnitude terms can make the difference in the two sides equal 0 even though the approximate solution is not accurate to the current digits of precision. Rationally factoring the difference in the two sides of an equation (see Section 4.6) before using the **soLve** command often results in more accurate solutions obtained more quickly.

Exercise: In exact mode, expand

$$(x - \pi)^4 = 0$$

Then in approximate mode with 6 digits of precision, solve both the expanded and factored forms of the equation. Finally compare the accuracy of the two approximate solutions with π (3.14159...), the exact answer.

If precision is in mixed mode, the **soLve** command first attempts to solve the equation for all of its real and complex solutions using the exact-mode algorithm. However, irrational subexpressions (such as π and $\sqrt{2}$) in the equation are approximated. Then if there are any subproblems that cannot be solved exactly and that contain only one variable, this command attempts to solve each subproblem approximately for one real solution using the approximate mode algorithm. It starts searching in the interval [−10, 10] and repeatedly doubles the interval until a solution is found or you press **Esc**.

Relations can have any number of solutions, including none. When ***DERIVE*** can find no solution, it sounds an error beep and displays the warning message

No solutions found

Exercise: Try to solve the equation

$$x = x + 1$$

for x.

When an equation or an inequality is degenerate, any value for the variable is a solution, and ***DERIVE*** returns a new variable of the form *@n* where n is an integer. (Think of "@" as an abbreviation for "arbitrary" or "anything".) n starts at 1 and increments by 1 each time a new variable is required. For example, solving the equation

$$2 \cdot x = x + x$$

for x yields

$$x = @1$$

Exercise: Solve the inequality

$$|x - 2| \geq 0$$

for x.

4.15 Solving Systems of Equations

Use the **soLve** command to directly solve a system of equations that is linear in the solution variables (the variables for which you want a solution). The equations can be nonlinear in variables other than the solution variables. The system of equations is entered in the form of a vector (see Section 8.1). If there are more variables than equations, the **soLve** command also prompts you to enter the solution variables.

For example, consider the following system of equations linear in x and y:

$$2 \cdot a^2 \cdot x + 3 \cdot y = 7$$
$$x - 5 \cdot y = 0$$

To make a vector of these equations, enter them on the author line, enclose them in square brackets, and separate them by a comma. This yields

$$[2 \cdot a^2 \cdot x + 3 \cdot y = 7,\ x - 5 \cdot y = 0]$$

Solving this vector of equations for x and y yields the solution vector

$$\left[x = \frac{35}{10 \cdot a^2 + 3},\ y = \frac{7}{10 \cdot a^2 + 3}\right]$$

A system of equations is **singular** if the equations are not independent of one another. If a singular system of linear equations is consistent, the solution set will contain one or more arbitrary values of the form *@n* where *n* is an integer. The number of distinct arbitrary values in the solution set is the number of solution variables minus the number of independent equations.

Exercise: Solve the equations

$$x + 3 \cdot y = 1$$
$$2 \cdot x + 6 \cdot y = 2$$

for x and y.

If a singular system of linear equations is inconsistent, the **soLve** command sounds an error beep and displays the warning message

No solutions found

Exercise: Attempt to solve the equations

$$x + 3 \cdot y = 1$$

$$2 \cdot x + 6 \cdot y = 3$$

for x and y.

The **soLve** command cannot directly solve a system of equations *non*linear in its solution variables. However, ***DERIVE*** can eliminate much of the error-prone drudgery involved in finding an exact solution to such a system by using the following procedure:

- If some subset of the equations is linear in some subset of the solution variables, use the **soLve** command to solve for those variables in those equations.
- Use the **Manage Substitute** command to substitute those solutions into the remaining equations, thus simultaneously eliminating the linearly occurring variables and obtaining a smaller system of equations.
- Use the **Build** command to try to obtain a triangular system of equations by linearly combining equations in order to annihilate polynomially occurring solution variables (i.e. by doing Gaussian elimination).
- When all solution variables but one have been annihilated from an equation, use the **Solve** command to solve this univariate equation for the final variable.
- Use the **Manage Substitute** command to back substitute all combinations of solutions in order to solve for the solution variables that were annihilated.

The univariate equations that result from this process will often be unsolvable using the **soLve** command in exact mode. If this occurs, plot the difference in the two sides of the equation to determine appropriate intervals. Then use the **soLve** command in approximate mode to find accurate solutions.

Functions are defined in SOLVE.MTH for computing an approximate real or complex solution to a nonlinear system of equations (see Section 9.1). Even if the above procedure does not reduce a nonlinear system all the way to univariate equations, the functions defined in SOLVE.MTH can be used on the reduced system to compute an approximate solution.

For example, consider the following system of nonlinear equations:

$$x^2 - z + w + 1 = 0$$

$$y^2 - z - w + 3 = 0$$

$$x^4 + 2\cdot x^2\cdot y^2 + 10\cdot x^2 + x + y^4 + 9\cdot y^2 - 2\cdot y - 4\cdot z^2 + 2\cdot w - 2 = 0$$

$$x^2 + 2\cdot x - y^2 + y + z - w - 23 = 0$$

Since the first two equations are linear in z and w, use the **soLve** command on them to find z and w in terms of x and y. This yields

$$z = \frac{x^2 + y^2 + 4}{2}$$

$$w = -\frac{x^2 - y^2 - 2}{2}$$

Next use the **Manage Substitute** command to replace z and w with the right sides of these equations in the last two of the original equations. This yields

$$x^2 + x + 2\cdot y^2 - 2\cdot y - 16 = 0$$

$$2\cdot x^2 + 2\cdot x - y^2 + y - 22 = 0$$

Since both of these equations are polynomial in x, use the **Build** command to subtract the second equation from two times the first equation to annihilate the highest power of x. This yields

$$5\cdot y^2 - 5\cdot y - 10 = 0$$

Fortuitously x was eliminated in a single step, but in general, it will take a number of linear combinations of equations to steadily reduce the degree of a variable to 0. Since the equation is univariate, use the **soLve** command to solve for y. This yields the solutions

$$y = -1$$

$$y = 2$$

Finally use the **Manage Substitute** command to back substitute each of these solutions for y into the appropriate equations to solve for x, z, and w. The following are the four solutions to the system of nonlinear equations:

$$(x=3,\ y=-1,\ z=7,\ w=-3)$$

$$(x=-4,\ y=-1,\ z=21/2,\ w=-13/2)$$

$$(x=3,\ y=2,\ z=17/2,\ w=-3/2)$$

and

$$(x=-4,\ y=2,\ z=12,\ w=-5)$$

To eliminate possible spurious solutions, each of these results should be substituted into the original equations.

4.16 Boolean Operators and Truth Tables

Boolean operators make it possible to logically combine relations and/or truth-valued variables. The following Boolean operators are built into ***DERIVE***:

- NOT (negation): NOT *p* simplifies to false if *p* is true, and to true if *p* is false.
- AND (conjunction): *p* AND *q* simplifies to true if both *p* and *q* are true, and to false if either *p* or *q* or both are false.
- OR (disjunction): *p* OR *q* simplifies to true if either *p* or *q* or both are true, and to false if both *p* and *q* are false.
- XOR (exclusive or): *p* XOR *q* simplifies to true if either *p* or *q* but not both are true, and to false if both *p* and *q* are false or if both *p* and *q* are true. If the truth values of *p* and *q* are unknown, it simplifies to the equivalent Boolean expression (NOT *p* AND *q*) OR (*p* AND NOT *q*).
- IMP (implication): *p* IMP *q* simplifies to true if *p* is false or *q* is true, and to false if *p* is true and *q* is false. If the truth values of *p* and *q* are unknown, it simplifies to the equivalent Boolean expression NOT *p* OR *q*.

NOT is a prefix operator and the other Boolean operators are infix operators. As with the numerical operators (see Section 3.1), you can use parentheses to control the order in which Boolean operators are applied.

In the absence of parentheses, Boolean operators with higher precedence are applied before those with lower precedence. The Boolean operators above are listed in order of decreasing precedence. For example, entering

```
NOT p OR q AND r
```

is equivalent to entering

```
(NOT p) OR (q AND r)
```

DERIVE uses the laws of Boolean algebra to simplify expressions involving Boolean operators into simpler ones having the same truth value. For example,

```
NOT NOT p
```

simplifies to `p`, and

```
p AND (p AND q OR r)
```

simplifies to

```
p AND (q OR r)
```

Relations combined by Boolean operators are also simplified when possible. For example,

```
6 ≥ -2·x AND 3·x /= -9
```

simplifies to

```
x > -3
```

The Boolean truth value constants are denoted by true and false. For example,

```
p OR NOT p
```

simplifies to true, and

```
NOT true
```

simplifies to false.

A truth table shows each of the truth values of a Boolean expression when its variables are assigned all possible permutations of truth values. Use the TRUTH_TABLE function to generate a truth table matrix for one or more Boolean expressions. If p_1, p_2, ... are truth variables,

TRUTH_TABLE(p_1, p_2, ..., $boolean_1$, $boolean_2$, ...)

generates a truth table matrix for the Boolean expressions. For example,

```
TRUTH_TABLE(p, q, p AND q, p OR q, p XOR q, p IMP q)
```

simplifies to

p	q	p AND q	p OR q	p XOR q	p IMP q
true	true	true	true	false	true
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

When given integer operands, the Boolean operators NOT, AND, OR, XOR, and IMP perform bit-wise logical operations on the integers. The operations are performed as if integers are stored using a "half-infinite" two's-complement representation. For example, 3 OR 5 simplifies to 7, and NOT 5 simplifies to –6.

Chapter 5. Plotting

A well-framed plot is perhaps the best way to get an overall "feel" for how an expression or function behaves. Section 5.1 describes how to make 2-dimensional explicit, implicit, and parametric plots in rectangular or polar coordinates. Section 5.2 describes how to make 3-dimensional wire-frame plots of bivariate expressions with and without hidden line removal. Note that the resolution of and colors available for these plots depend on the display mode (see Section 2.12).

Functions for plotting space curves, parametric surfaces, and complex-valued expressions are defined in the utility file GRAPHICS.MTH (see Section 9.19).

5.1 2D-plot Window

DERIVE's 2D-plot window is used for plotting the following types of expressions and equations generated in an algebra window:

- expressions involving a single variable (i.e. univariate expressions);
- equations of the form $y = u$, where y is a variable and u is a univariate expression in another variable; and
- equations of the form $u = v$, where u and v are expressions of at most two variables (see Section 5.1.9).

Section 5.1.1 describes how to plot a function *explicitly* defined by a univariate expression using rectangular coordinates. Section 5.1.2 describes how to set and change plot window preferences. Section 5.1.3 describes how to optimally frame plots. Section 5.1.4 describes how to plot multiple expressions. Section 5.1.5 describes how to optimize the speed and accuracy at which expressions are plotted. Section 5.1.6 describes how to plot an expression using polar coordinates. Section 5.1.7 describes how to make a parametric plot of a pair of expressions. Section 5.1.8 describes how to plot data matrices. Section 5.1.9 describes how to plot a function *implicitly* defined by an equation involving two variables.

5.1.1 Plotting Univariate Expressions

Use the following procedure to plot a univariate expression, subexpression, or equation whose right side is a univariate expression:

- Issue the algebra window's **Plot** command. It displays the selection field

  PLOT: Beside Under Overlay

  and waits for you to choose the location for the plot window relative to the algebra window. If you choose Beside or Under, you are prompted for the column or row at which to make the split (see Section 2.13).

  Note that overlaying plots provides more room for both expressions in the algebra window and plots in the plot window. Plotting beside or under the algebra window allows expressions and plots to appear simultaneously, and they do not have to be redisplayed when you switch windows.

- For now, choose Overlay. This opens a 2D-plot window that covers the algebra window. Note that the phrase

  Derive 2D-plot

  is displayed at the right end of the status line, indicating that a 2D-plot window is currently active.

- Issue the 2D-plot window's **Plot** command to actually plot the highlighted expression. At the left end of the message line, a phrase is displayed indicating the expression number and plot color. This message will be displayed until plotting is completed. If you wish to stop the plotting before completion, press the **Esc** key.

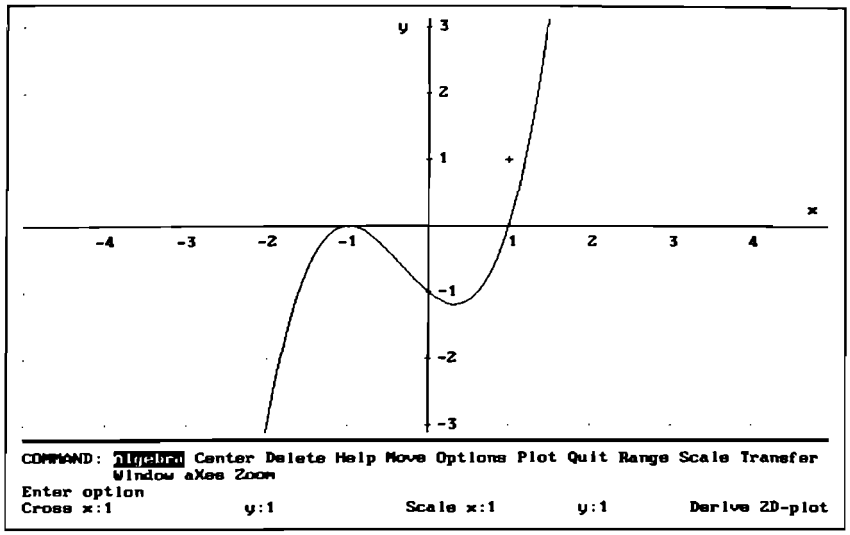
- On PC compatible computers, to send a graphics image of a ***DERIVE*** screen including a 2D-plot window to a printer, press **Shift-F10**. To send just the plot *without* the menus and status line at the bottom of the screen, press **Shift-F9**. Before printing you must select the appropriate printer type using the **Transfer Print Options** command (see Sections 2.10 and 2.11).

Exercise: Plot either the following expression or equation:

$$(x - 1) \cdot (x + 1)^2$$

$$y = (x - 1) \cdot (x + 1)^2$$

The screen should look similar to:



A 2D Rectangular Plot

If your screen does not look like this, then ***DERIVE*** may be in text mode. If your computer has a graphics display adapter, use the 2D-plot window's **Options Display** command or press the **F5** function key to switch to graphics display mode (see Section 2.12). When you change display modes, the expression is automatically replotted.

The plot's horizontal axis (titled x) represents the variable values and the vertical axis (titled y) represents the corresponding expression values.

In the 2D-plot window there is a movable **cross** whose coordinates are displayed on the status line at the bottom of the screen. Note that x is the name of the horizontal coordinate of the cross and y is the name of the vertical coordinate, regardless of the name used for the variable in the expression being plotted.

Use the 2D-plot window's **Move** command to display and/or change the current horizontal and vertical coordinates of the cross. The coordinates are displayed using the current notation style (see Section 3.9). Alternatively, you can move the cross by using the direction keys as follows:

- **Up arrow** or **Ctrl-E** moves the cross up one pixel.
- **Down arrow** or **Ctrl-X** moves the cross down one pixel.

- **Left arrow** or **Ctrl-S** moves the cross left one pixel.
- **Right arrow** or **Ctrl-D** moves the cross right one pixel.
- **PgUp** or **Ctrl-R** moves the cross up half a tick mark.
- **PgDn** or **Ctrl-C** moves the cross down half a tick mark.
- **Ctrl-Left arrow** or **Ctrl-A** moves the cross left half a tick mark.
- **Ctrl-Right arrow** or **Ctrl-F** moves the cross right half a tick mark.

When you move the cross, note how the cross coordinates change on the status line. Because of limited space, only a few digits of the coordinates are displayed. To more accurately display the cross position, issue a **Move** command and examine the coordinate values. To increase the number of digits displayed by the **Move** command, issue an **Options Notation** command and set the desired number of digits. To horizontally scroll the digits displayed by the **Move** command, press the **End** key or **Ctrl-Q D** to move the cursor to the right end of the field.

If you move the cross to a location that appears to be "on" a plot, it may be near but not quite on the plot. To more accurately display the coordinate values of a specific point on the plot, switch to **trace mode** by pressing the **F3** function key. Alternatively, you may turn on trace mode by issuing the **Options State** command and selecting Yes in the Trace field. In trace mode, the cross is confined to moving along (or tracing) the plot from point to point while displaying the corresponding coordinate values on the status line.

When you switch to trace mode the cross will move up or down to the plot line at the current x coordinate. In trace mode the cross displays as a small square box, whose center lies on the plot line. The expression number of the plot being traced is displayed on the message line. This makes trace mode useful for distinguishing plot lines when plotting multiple expressions (see Section 5.1.4).

Once in trace mode, use the direction keys to move the box along the plot as follows:

- **Left arrow** or **Ctrl-S** moves the box left one pixel and up or down as necessary to stay on the plot.
- **Right arrow** or **Ctrl-D** moves the box right one pixel and up or down as necessary to stay on the plot.
- **Ctrl-Left arrow** or **Ctrl-A** moves the box left half a tick mark and up or down as necessary to stay on the plot.

- **Ctrl-Right arrow** or **Ctrl-F** moves the box right half a tick mark and up or down as necessary to stay on the plot.

The **Up arrow** and **Down arrow** keys are used to trace another plot line in a 2D-plot window with multiple expressions plotted (see Section 5.1.4). The **PgUp** and **PgDn** keys are disabled in trace mode.

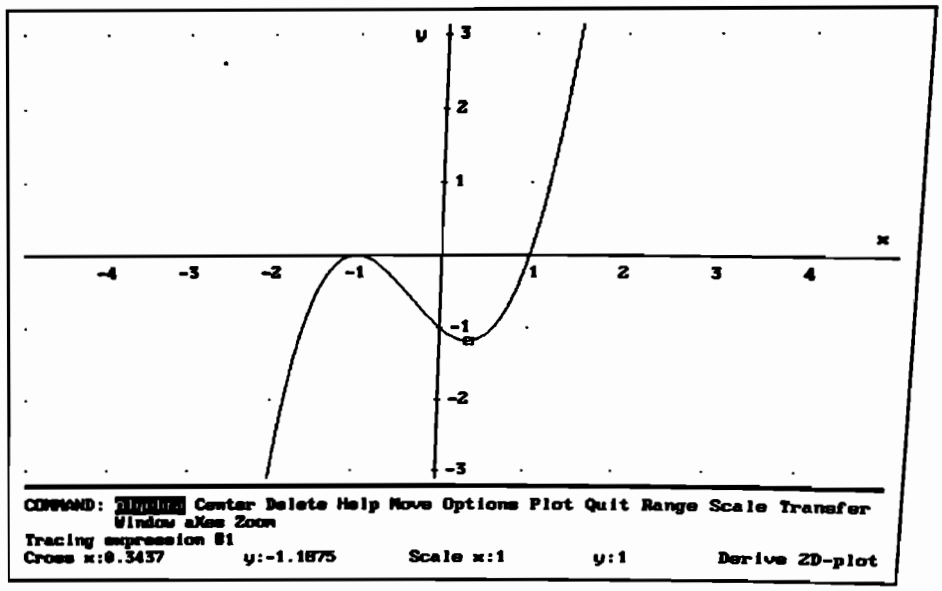
In trace mode the **Move** command moves the box to the point specified by the horizontal and vertical cross coordinates, regardless of whether this point is *on* the plot line. To move the box back to the plot line and continue tracing, use the direction keys as explained previously.

To switch off trace mode, press the **F3** function key again. Alternatively, you may turn off trace mode by issuing the **Options State** command and selecting No in the Trace field. The cross will return to its original shape and the cross direction keys to their original use.

Tracing is only available in graphics display mode (see Section 2.12). All 2D plots may be traced except for data matrix and implicit plots. If you try to turn on trace mode, hear an error beep, and the cross does not move to the plot line, either there is no explicit plot to trace or the expression is not real and finite at the cross's current x coordinate. Move the cross left or right to where the expression is real and finite and try again to switch to trace mode.

Tracing a plot is especially helpful in visually locating intercepts and minimum and maximum expression values.

Exercise: Using the same plot as in the previous exercise, press **F3** to turn on trace mode. Use the cross direction keys to trace along the plot and confirm two x-intercepts at $x=-1$ and $x=1$ and one y-intercept at $y=-1$. Use tracing to find the coordinates of the plot's local minimum. The screen should look similar to:



The Local Minimum

Finally, press **F3** to turn off trace mode and use the **Window Close** command (see Section 2.13) to close the 2D-plot window.

If, while tracing along a plot, you encounter a point at which the expression does not have a finite real value, the message

Tracing expression *#n*, not real and finite!

is displayed and a warning error beep occurs. This makes tracing useful for locating removable singularities that may not be apparent on the plot (see Section 4.8). If you continue moving the cross, the message will remain until the expression has a finite, real value.

Exercise: Plot the expression

$$\frac{\text{SIN}\cdot(x)}{x}$$

Turn on trace mode and trace along the plot to locate the removable singularity at $x=0$. Notice the warning error beep and message line. Continue tracing past the removable singularity. Turn off trace mode.

For a collection of univariate expressions that are interesting to plot, load the file PLOT2D.MTH using the algebra window's **Transfer Merge** command (see Section 2.9).

5.1.2 Setting 2D-plot Window Preferences

Use the commands described in this section to customize the plot and axes colors, axes titles and labels, and tick marks.

The plot window is initially displayed with horizontal and vertical axes, titles, numbered labels, and tick marks. The axes components may be removed from the plot window display in their entirety by issuing the **aXes** command and selecting not to display the axes. Additionally, the display of the axes components may be changed individually by issuing the **aXes** command and responding to the appropriate prompt. To display different titles for the horizontal and vertical axes (other than the default *x* and *y*), simply type in the desired title(s). If no title is desired, delete the existing title and leave the prompt blank. Axes labels may be numbered or left blank. If they are numbered, ***DERIVE*** puts as many on the screen as possible, using scientific notation when appropriate.

Use the Rows and Columns fields of the **aXes** command to adjust the number of character rows and/or columns per tick mark. This will in turn change the **aspect ratio**. The aspect ratio of a 2D-plot window is the ratio of the vertical distance between tick marks to the horizontal distance between tick marks. If the aspect ratio is 1 *and* the plot scale is the same in both directions, circles will plot as circles rather than ellipses. The status line shows the current plot scales in units per tick mark. The numbers following the *x* and *y* give the horizontal and vertical scale, respectively. Since computer screens vary widely, the initial aspect ratio may not be optimum for your computer.

Use the **Options Color Plot** command to change the color of the axes, the cross, and the *next* expression that is plotted. This command displays the integers 0 through 15 on the message line, each number displayed in its respective color, and prompts you to enter a color number. You can enter any integer from 0 to 15, however the actual number of colors available depends on your computer's graphics hardware and the current display mode (see Section 2.12). Note that the axes labels, titles and tick marks will be displayed in the same color as the axes themselves. The color of the last expression plotted can be changed by reissuing a **Plot** command; successive plots cycle through all the available colors unless an **Options Color Auto** command is used to turn off this feature.

Exercise: Plot the following expressions:

$$\sqrt{(4 - x^2)}$$

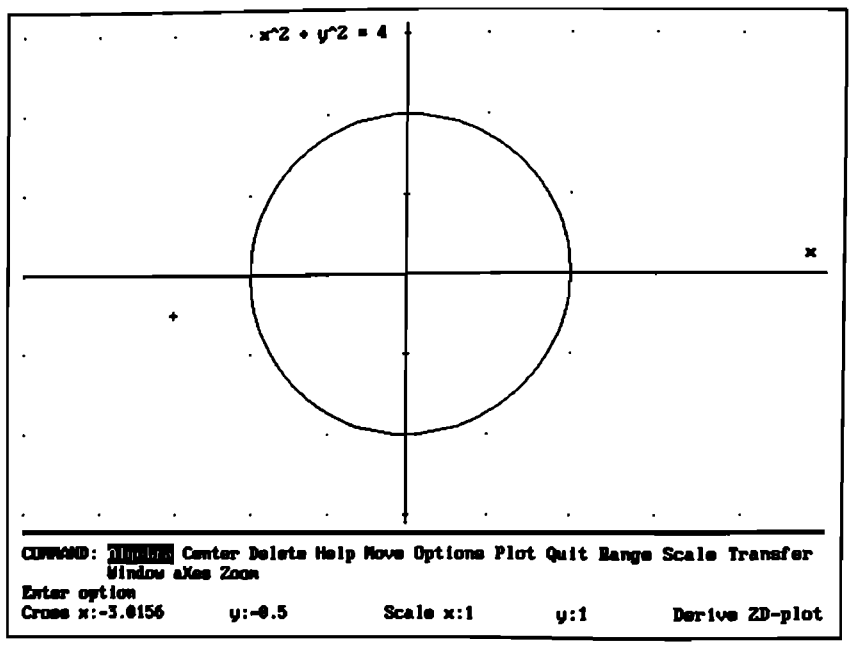
$$-\sqrt{(4 - x^2)}$$

Then use the **aXes** command to adjust the aspect ratio so the expressions plot as a circle rather than an ellipse. Use the **Transfer Save State** command to save this aspect ratio in the *DERIVE* initialization file.

Now, use the **aXes** command to change the title of the vertical axis from `y` to `x^2 + y^2 = 4` and also to turn off the numbering of the axes labels.

If in a display mode having multiple colors (see Section 2.12), use the **Options Color Auto** and **Options Color Plot** commands to change the plot color of the last expression plotted so that it is the same as the first expression plotted. Both halves of the circle should now be plotted in the same color.

The screen should look similar to:


x^2 + y^2 = 4
x
COMMAND: Center Delete Help Move Options Plot Quit Range Scale Transfer Window aXes Zoom
Enter option
Cross x:-3.0156 y:-0.5 Scale x:1 y:1 Derive 2D-plot

A "Square" Circle

Note that an implicit plot of the equation:

$$x^2 + y^2 = 4$$

also generates a circle of radius 2 about the origin (see Section 5.1.9).

5.1.3 Framing 2D Plots

A major objective of plotting is to reveal the qualitative features of an expression. For example, you usually want a plot to reveal the locations of all maxima, minima, inflections, zeros, poles, discontinuities and stationary points as well as any symmetries, periodicity, and asymptotic behavior. Consequently, it is advantageous to optimally frame the plot so that these points of interest are visible in the 2D-plot window.

Use the commands described in this section in conjunction with the cross to optimally frame a plot. These commands are: **Zoom**, **Scale**, **Center**, and **Range**. The **Zoom** and **Scale** commands change the magnification of the plot and automatically adjust the window borders. The **Center** command adjusts the borders of the window based on the cross position. The **Range** command allows direct specification of the window borders and automatically adjusts the plot magnification.

The general procedure for optimally framing a plot is to position the 2D-plot window in the middle of the points of interest. Then zoom in or out as necessary to magnify the plot as much as possible while keeping all the interesting points visible in the window.

Note that if a plot is poorly framed, you can issue additional 2D-plot window commands *before* the plot is complete. However, if you want to abort a plot before issuing a command, press the **Esc** key.

Use the **Zoom** command to change the plot scale. It displays a multi-field line similar to

ZOOM: Axis: Both X Y Direction: (In) Out

Zooming in magnifies the plot and zooming out displays a greater portion of the plot. You can zoom in the horizontal direction, the vertical direction, or both directions at once. When you zoom in or out, the plot scale changes to the next closest number in the sequence: ... 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20 ... . Recall that the status line displays the current plot scales in the horizontal (denoted by x) and vertical (denoted by y) direction in units per tick mark.

The function keys provide an alternative to using the **Zoom** command. The following function keys decrease the plot range, thus magnifying the plot:

- **F7** zooms the vertical direction in.
- **Shift-F7** zooms the horizontal direction in.
- **F9** zooms both directions in.

The following function keys increase the plot range, thus displaying a greater portion of the plot:

- **F8** zooms the vertical direction out.
- **Shift-F8** zooms the horizontal direction out.
- **F10** zooms both directions out.

You do *not* have to wait for a plot to complete before selecting a menu option or pressing a function key. For example, to zoom both directions in by a factor of ten, press **F9** three times in quick succession.

Use the **Scale** command to display and/or change the plot scale to exactly the value you want. Change the x value to adjust the scale in the horizontal direction and the y value to adjust the scale in the vertical direction. This command gives more precise control over the scales than the **Zoom** command but requires more keystrokes. Also, the simple decimal number scale values given by the **Zoom** command make it easy to estimate coordinate values from tick marks and/or axes labels without having to use the cross.

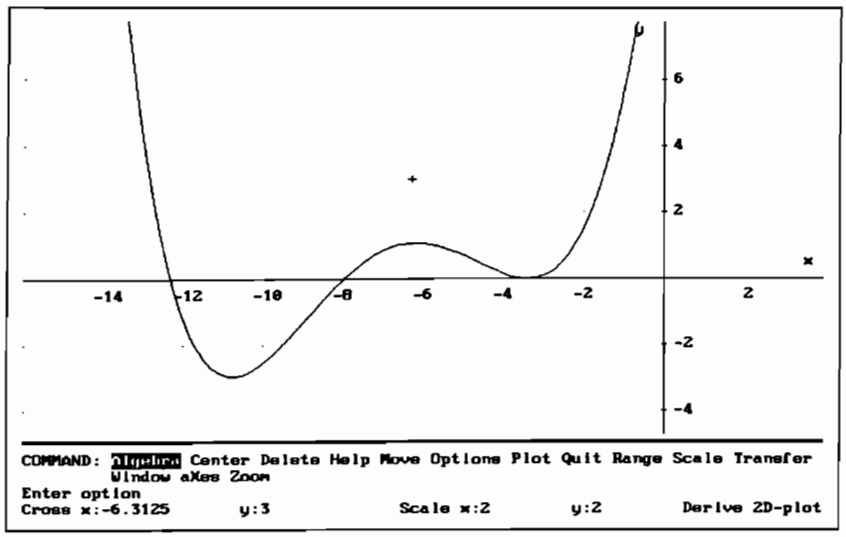
Initially the axes origin is at the center of the 2D-plot window. Use the **Center** command to position the center of the window at the cross. Thus to frame an interesting part of a plot, move the cross to the point of interest (by using the cross direction keys or the **Move** command) and then issue a **Center** command.

Exercise: Plot and optimally frame the quartic polynomial

$$\frac{x^4}{80} + \frac{407 \cdot x^3}{1200} + \frac{18637 \cdot x^2}{6000} + \frac{67949 \cdot x}{6000} + \frac{10693}{750}$$

Hint: Zoom out and move the frame left relative to the plot line.

The optimally framed plot should look similar to:



A Framed Quartic Polynomial Plot

Finally, turn on trace mode and trace along the plot to determine the expression's maxima, minima, and zeros.

You can use the **Windows Designate** command to quickly delete the current plot(s) and restore the initial default plot settings for plot scaling, center, accuracy, etc. Just press **Enter** to select the 2D-plot window when the command prompts you for a window type (see Section 2.13).

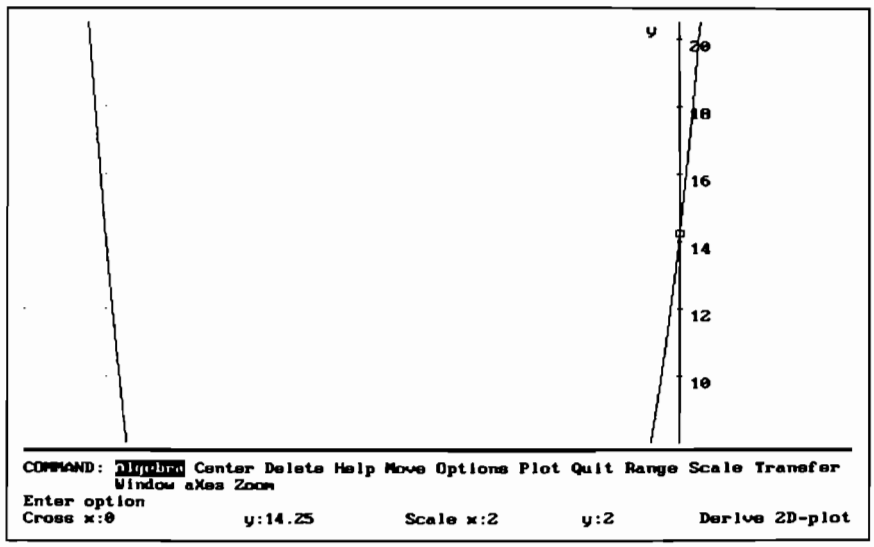
If you are interested in a part of the plot not visible in the 2D-plot window, you may want to switch to **follow cross mode**. In this mode, the window "follows" the cross when it is moved outside the window borders. Note that the cross may be moved outside the window borders as a result of a command (e.g. **Move**, **Zoom**, or **Scale**) or by using the cross direction keys. To turn on follow cross mode, issue the command **Options State** and select Yes in the Follow field. Initially, follow cross mode is on, unless changed in the ***DERIVE*** initialization file (see Section 2.14).

In follow cross mode, if the cross is moved just outside a window border, the window will be scrolled by 1/3 of the window dimension in the appropriate direction. If the cross is moved far from any window border, the window will be centered on the new cross position.

It may be advantageous to turn on follow cross mode when tracing a plot (see Section 5.1.2). If follow cross mode is off and you trace to a point on the plot that

extends past the window borders the cross will disappear from sight. However, the coordinate values of the cross will continue to be updated on the status line. If follow cross mode is on while you are tracing a plot, the window will be scrolled to follow the cross as it moves outside the 2D-plot window.

Exercise: Use the **Options State** command to turn off follow cross mode if it is currently on. Using the same plot as in the previous exercise, trace along the plot to the *y*-intercept (at 14.25). Note that the cross has disappeared. Use the **Options State** command to turn on follow cross mode. Note that the window will be scrolled so that the cross at its current location (0, 14.25) is now visible. The screen should look similar to:



A Framed *y*-Intercept

Finally, turn off trace mode and use the **PgDn** key to move the cross down, scrolling the window until the *x* axis is visible.

You may prefer to directly specify the borders of the 2D-plot window rather than zooming in and centering on a particular portion of a plot. Use the **Range** command to display and/or adjust the plot range either by entering new border coordinates or by positioning the range box. This command displays a multi-field line similar to

RANGE: Left: -5 Right: 5 Bottom: -3.125 Top: 3.125

The boundaries of the plot range are determined by the left, right, bottom, and top borders of the 2D-plot window whose coordinates correspond to the Left, Right,

Bottom, and Top field values. The plot range is also depicted visually in the 2D-plot window by means of a range box. The border coordinates may be changed by typing in a new value. Modifications to a border coordinate will be reflected in the range box when you move the cursor to a different field. Press **Tab** to move the cursor to the next field, **Shift-Tab** to move the cursor to the previous field.

An alternative to typing in a new border coordinate is to use the direction keys to position the range box over the desired plot range. If the cursor is in the Left or Top fields, then the direction keys will adjust the left and top borders of the range box. Similarly, if the cursor is in the Right or Bottom fields, then the direction keys will adjust the right and bottom borders of the range box.

Use the direction keys to position the range box as follows:

- **Left arrow** moves the Left or Right border left one pixel.
- **Right arrow** moves the Left or Right border right one pixel.
- **Ctrl-Left arrow** moves the Left or Right border left half a tick mark.
- **Ctrl-Right arrow** moves the Left or Right border right half a tick mark.
- **Up arrow** moves the Top or Bottom border up one pixel.
- **Down arrow** moves the Top or Bottom border down one pixel.
- **PgUp** moves the Top or Bottom border up half a tick mark.
- **PgDn** moves the Top or Bottom border down half a tick mark.

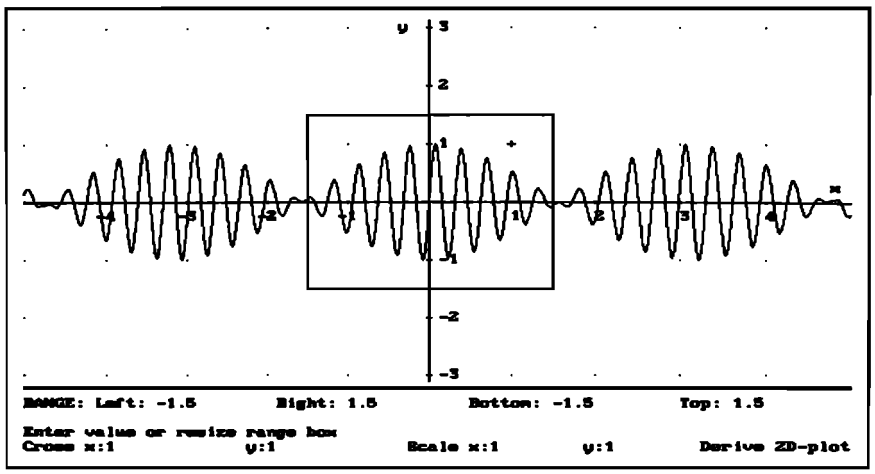
The left and right arrow keys have a dual use. Initially, they are used to position the range box. Note that the range box is only displayed when ***DERIVE*** is in graphics display mode (see Section 2.12). If you press the **F6** function key, these keys move the cursor while editing a border coordinate. Press **F6** again to use the left and right arrow keys to adjust the borders of the range box.

You may press **Esc** at anytime to exit the **Range** command without changing the plot range. Press **Enter** to accept the new plot range as defined by the border coordinates and range box. ***DERIVE*** uses the new plot range to calculate the plot scale, changing it to the closest number in the scale sequence (see the **Zoom** command). The expression(s) will be replotted using the new plot range and scale. In order to maintain a simple scale it is sometimes necessary to make slight adjustments to the plot range so do not be alarmed if the border coordinates are not exactly the ones you entered.

Exercise: Plot the expression

$$\cos(x) \cdot \sin(20 \cdot x)$$

Issue the **Range** command. Use the direction keys and/or edit the border coordinates to position the range box over the area 1.5 units from the origin in each direction. The screen should look similar to:



The Range Box in Action

Now press **Enter** and the expression will be replotted using the new plot range and calculated scale. Issue the **Range** command to view the border coordinates. Press **Esc** to exit without making any changes. Finally, use the **Windows Designate** command and select the 2D-plot window to restore the default plot settings.

Auto-scale mode is also helpful in framing a plot. Before plotting an expression in this mode, it is evaluated at several sample points to estimate the range of values the expression assumes over the plot interval. This estimate is then used to adjust the vertical plot range so the plot will be reasonably framed. Note that: 1) some parts of an auto-scaled plot may lie outside the vertical plot range because only a sampling of data points is taken; 2) auto-scaling adjusts both the vertical plot scale and center to achieve the desired vertical plot range; and 3) auto-scaling does *not* adjust the horizontal plot range.

To turn on auto-scale mode, issue the **Scale** command and select **Yes** in the **Auto** field to automatically scale in the vertical direction. Any univariate expression plotted in rectangular coordinates *after* auto-scale mode is turned on will be automatically scaled. Auto-scale mode may be turned off by issuing the **Scale** command and

selecting No in the Auto field. Initially auto-scale mode is off, unless changed in the ***DERIVE*** initialization file (see Section 2.14).

If, for each variable value on the horizontal axis, the value of the expression lies above or below the 2D-plot window or is not a real number, no plot points will appear on the screen. If this happens, use one or all three methods described below to try and find plot points:

- Zoom out to "find" the plot line so it can be optimally framed.
- Turn on auto-scale mode and replot the expression.
- Switch back to the algebra window and try to find variable values for which the expression is real-valued using **Manage Substitute** and **approX** commands. Then return to the 2D-plot window and use the framing commands described in this section to insure that one or more of these points is visible.

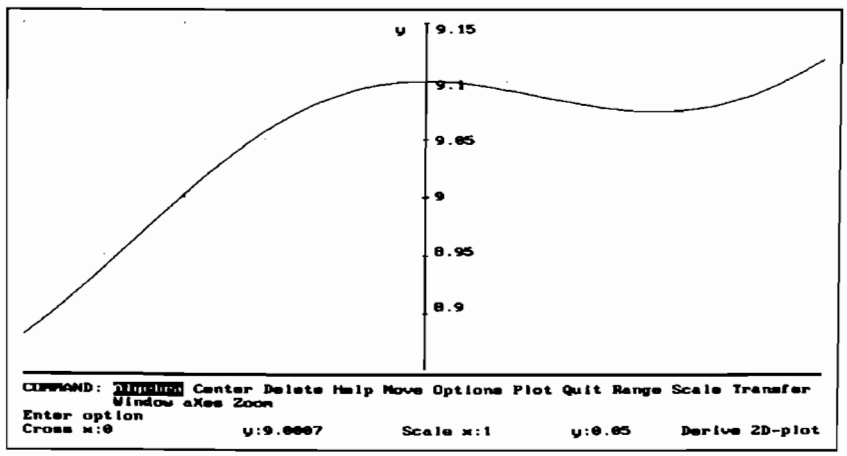
Exercise: Using the default plot range, plot the following expression

$$9 + 0.2\ \mathrm{SIN}(0.2\ x)^3 + 0.1\ \mathrm{COS}(0.3\ x)^2$$

Do you see any plot points?

> **Hint**: The correct answer is No because the plot line is outside of the default plot range.

Now, issue the **Scale** command and select Yes for automatic *y* scaling. Replot the expression by issuing a **Plot** command. The screen should look similar to:



An Auto-scaled Plot

Zoom out in both directions three times so that more of the plot line is revealed. Finally, turn off auto-scale mode by issuing the **Scale** command and selecting No in the Auto field.

Finding the optimal framing for a plot is a learning process. If you do not know in advance what values to enter for the **Scale** and **Move** commands, use the **Zoom**, **Center** and **Range** commands along with the direction keys and follow cross mode to get close to the desired framing. Then use all of the above together with the **Scale** command to fine-tune.

With experience, inspecting alternative forms of an expression can suggest the locations of interesting features, thereby aiding your framing explorations. For example, factored forms suggest the locations of zeros and poles, whereas expanded forms suggest the locations and strengths of poles together with asymptotic behavior.

If $F(x)$ is a user-defined function (see Section 4.12), you can apply various transformations to the function to distort the vertical and/or horizontal coordinates of a plot of $F(x)$:

- $LN(F(x))$ and $F(EXP(x))$ produce semi-log plots of $F(x)$.
- $LN(F(EXP(x)))$ produces a log-log plot of $F(x)$.

Exercise: For several positive values of n, make semi-log plots of

$$F(x) := n^x$$

Exercise: For several values of n, make log-log plots of

$$F(x) := x^n$$

5.1.4 Multiple Plots

To plot multiple expressions in the same 2D-plot window:

- Enter and plot the first expression (see Section 5.1.1).
- Use the 2D-plot window's **Algebra** command to switch back to the algebra window.
- Highlight a different expression or subexpression, using an **Author** or **Build** command to create it, if necessary.
- Use the algebra window's **Plot** command to switch back to the 2D-plot window.
- Use the 2D-plot window's **Plot** command to plot the new expression.

A vector of two expressions will be plotted as a parametric plot (see Section 5.1.7). However, if a vector of three or more expressions is highlighted, the **Plot** command plots each element of the vector. After entering each expression in an algebra window, use the **Declare vectoR** command or the **Author** command to make a vector of the expressions (see Section 8.1). Alternatively, the VECTOR function can be used to generate a vector of the values of an expression that is evaluated at a sequence of points (see Section 8.2). For example, simplifying the expression

```
VECTOR(x^k, k, 0, 2, 1/4)
```

gives

$$[1,\ x^{1/4},\ x^{1/2},\ x^{3/4},\ x,\ x^{5/4},\ x^{3/2},\ x^{7/4},\ x^{2}]$$

which, when plotted, displays as a family of curves in the 2D-plot window.

Note that multiple plots all share the same plot scale and range. Therefore, you may have to zoom out and/or move the center of the 2D-plot window to see all of the plot lines simultaneously.

To move a plot line relative to the others, add an offset to the corresponding expression in the algebra window. To magnify or shrink a plot line relative to the others, multiply the corresponding expression by a scaling factor in the algebra window. Sometimes it is advisable both to shift and scale plot lines in order to optimally frame several of them in a single window.

Exercise: Assume

$$5 + \mathrm{SIN}\left(\frac{t}{8}\right)$$

gives the velocity of a point at time t. Then

$$\frac{1}{8} \cdot \cos\left(\frac{t}{8}\right)$$

gives its acceleration. Shift and scale these expressions to optimally frame them in a single window.

Instead of individually scaling expressions, expressions may be plotted in multiple plot windows, each with their own scales.

Each 2D-plot window maintains a list of expressions to plot. An expression is added to the list if it is the highlighted expression in the algebra window when you use the **Plot** command in the 2D-plot window. Use the **Delete** commands to delete expressions from the list as follows:

- **Delete First** deletes the first expression in the list.
- **Delete Last** deletes the last expression in the list.
- **Delete Butlast** deletes all but the last expression in the list.
- **Delete All** deletes all the expressions in the list.

Deleting an expression from a 2D-plot window does *not* remove it from an algebra window. Similarly, removing an expression from an algebra window does *not* delete it from a 2D-plot window.

Expressions are plotted in the 2D-plot window in the same order as they were added to the plot list. If the expression being plotted is not of interest you may press **Esc** to skip to the next one in the plot list.

To compare coordinate values of multiple plot lines you may want to turn on trace mode (see Section 5.1.1). When you switch to trace mode the cross (displayed as a small box) will move to the plot line of the last expression in the list at the current x coordinate. You may then use the right and left arrow keys as described previously to trace along the plot. To trace another plot line use the up and down arrow keys as follows:

- **Up arrow** or **Ctrl-E** moves the box to the previous expression's plot line.
- **Down arrow** or **Ctrl-X** moves the box to the next expression's plot line.

In trace mode the number of the expression being traced is displayed on the message line. Thus when plotting several expressions, trace mode is a good way to determine the correlation between plot line and expression.

In a display mode with multiple colors (see Section 5.1), successive plots will automatically cycle through the available colors. Use the **Options Color Auto** and **Options Color Plot** commands to customize the colors used for multiple plot lines. (see Section 5.1.2).

5.1.5 Plot Speed and Accuracy

Use the **Options Accuracy** command to examine and/or change the plot accuracy used for plotting expressions. Generally, the higher the accuracy, the slower the plotting speed. You can set the plot accuracy to an integer between 0 and 9 inclusive, with 0 being the least accurate and 9 being the most accurate.

Normally ***DERIVE*** uses linear interpolation to greatly improve plotting speed. However, if the plot accuracy is 9, the expression being plotted is evaluated at every pixel column, and interpolation is *not* used. Use an accuracy of 9 if the expression may have narrow isolated spikes or bumps at some points in the plot interval.

Exercise: Using the initial plot accuracy of 7, plot the expression

$$\frac{1}{1 + 5000 \cdot (x - 0.5)^2}$$

Then change the plot accuracy to 9 and replot the expression. Finally, restore the plot accuracy to 7 and try zooming in to reveal details of the spike at $x=0.5$.

Because of interpolation, occasionally a vertical line may be plotted across a discontinuity. Use a plot accuracy of 9 to insure against the plotting of such vertical lines.

If the plot accuracy is less than 9, the expression is plotted as a sequence of straight line segments: ***DERIVE*** recursively subdivides the width of each plot interval. If n is the plot accuracy, the expression plots only at subinterval end points until a piecewise linear interpolation across the subintervals would be less than $8-n$ pixels from a straight-line interpolation across the subintervals. Thus, short intervals tend to be used where the curvature is high, and long intervals tend to be used where the curvature is low.

This adaptive plotting algorithm is usually much faster than evaluating the expression at every pixel column. Since the algorithm plots points throughout the plot interval before filling in details, it often becomes apparent that you should zoom, scale, or center before a plot is complete. Doing so may save you even more time. Experiment to determine the minimum plot accuracy that is sufficient for the types of expressions you are plotting.

If the value of the expression at either end point of the plot interval is not finite and real, the interval is subdivided until the end points are at adjacent pixel columns. Thus, plotting an expression is slow if the interval being plotted includes values where the expression is infinite or imaginary.

For high-degree polynomials and many other expressions, the second derivative often has large magnitudes where the expression has large magnitudes. For such expressions, extensive subdivision often occurs for portions of the plot line that are above or below the window.

For this reason, plotting is often slowest while nothing new is appearing inside the window! If nothing new appears on the screen for ten seconds or so, zoom *out* enough to get an overview of the plot line. Then use centering, zooming, and scaling to optimally frame (see Section 5.1.3) the portions of interest in the window.

Approximate mode arithmetic is always used while plotting (see Section 3.8). Initially, the digits of precision is that set by the last **Options Precision** command. As you zoom in, ***DERIVE*** automatically increases the precision enough to insure that the x and y coordinates of neighboring pixels are represented by unique numbers. However, catastrophic cancellation or sufficient accumulation of roundoff errors while evaluating the expression may cause irregularities on a plot line that should be smooth. If you notice this behavior, use the **Options Precision** command to increase the precision digits until the irregularity disappears, or you become convinced that it is a true feature of the plot.

Plot speed and accuracy can depend greatly upon the form of an expression because ***DERIVE*** does not simplify an expression before evaluating it for plotting. For example, when plotting a univariate expression, for each sample coordinate value that is substituted for the variable in the expression, the expression value is calculated using rational approximations and rounded rational arithmetic.

It is impractical to give concise universal advice about what forms of expressions plot faster and with more accuracy. However, rational factoring often produces the fastest and most accurate form. Unfortunately, rational factoring can require a prohibitive amount of time, more than you might save in plotting the resulting form. In such cases, you might instead want to try the less ambitious squarefree factoring. If that is also taking too long, then try the least ambitious trivial factoring. However, trivial factoring or even simplifying an expression might require a prohibitive amount of time when simplification entails extensive expansion. Thus, an unsimplified or partially simplified expression might be most practical.

Expansion involves a preliminary simplification, so expanding an expression generally takes more time than simplifying it. Moreover, expanded polynomials are often much more sensitive to roundoff error than are rationally factored polynomials or forms intermediate between these.

Although rational factoring is often helpful for plot speed and accuracy, the more extensive radical factoring usually decreases speed and accuracy. Radically factored forms usually have higher operation counts, and some of these operations are relatively time-consuming approximations to square roots, cube roots, trigonometric, or inverse trigonometric functions. Moreover, the underlying quadratic, cubic and quartic formulas can suffer catastrophic cancellation.

Complex factoring is even less advisable for plotting because in addition to the evils of radical factoring, complex factoring can introduce complex numbers whose imaginary parts ought to cancel but do not quite cancel because of roundoff error. Such bogus imaginary parts can unnecessarily suppress portions of the plot. You can remove these imaginary parts by using the RE function to extract the real part, but it is better to avoid generating them if possible.

Exercise: Using a precision of 6 digits and a plot accuracy of 7, compare the actual computing times to plot the factored and expanded forms of $(x-1)^9$. Finally, zoom in sufficiently to reveal dramatic differences between the two plot lines.

For additional plot speed, polynomials can be edited into a nested Hornerized form that avoids some recomputation of powers. For example, you can use either the **Build** or **Author** command together with **F3** and line-editing to convert

$$1024\cdot x^{11} - 2816\cdot x^{9} + 2816\cdot x^{7} - 1232\cdot x^{5} + 220\cdot x^{3} - 11\cdot x$$

into

$$((((1024\cdot x^{2} - 2816)\cdot x^{2} + 2816)\cdot x^{2} - 1232)\cdot x^{2} + 220)\cdot x^{2} - 11)\cdot x$$

Note that the coefficients and occurrences of x are unchanged; we have merely inserted parentheses and reduced exponents to their differences.

Hornerized form is often more accurate too because any cancellations tend to occur earlier on operands that are less contaminated with roundoff error.

Combining the above techniques, you can rationally factor a univariate polynomial, then Hornerize the resulting factors to obtain a particularly efficient form.

The choice between alternate forms is less certain for rational expressions. Expansion of rational expressions yields a partial fraction expansion with respect to the rationally factored common denominator. Sometimes an expanded or partially expanded form is faster and/or more accurate than factored forms. Moreover, sometimes a continued fraction form is faster and/or more accurate than either a factored or expanded form. If your rational expression exhibits unacceptable roundoff error, try one of the other alternatives.

5.1.6 Polar Plots

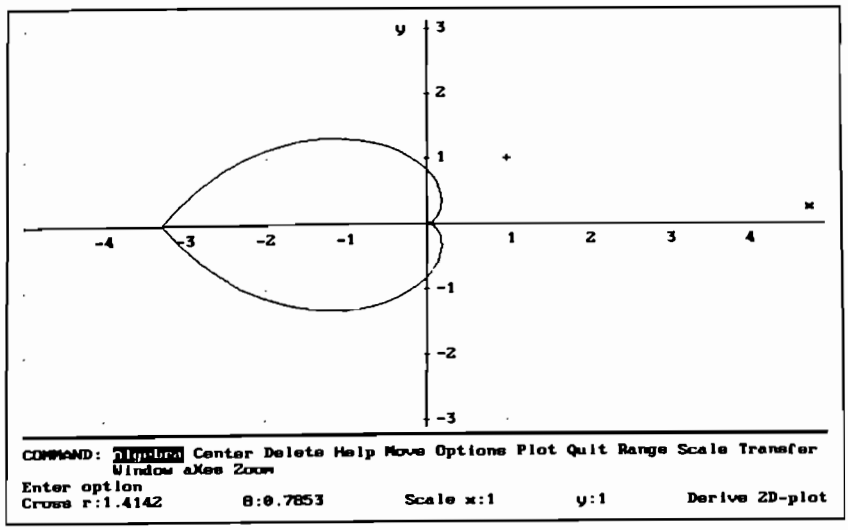
If a univariate expression is plotted in **rectangular coordinates**, the value of the variable is plotted as the horizontal distance to the right of the *y*-axis. The value of the expression is plotted as the vertical distance above the *x*-axis.

If a univariate expression is plotted in **polar coordinates**, the value of the variable is plotted as the angle measured counterclockwise from the positive *x*-axis. The angle may be represented in radians or degrees depending on the angle mode set by the algebra window's **Manage Trigonometry** command (see Section 6.3). The value of the expression is plotted as the radial distance from the origin. Negative values are plotted opposite the positive direction.

Use the Coordinates field of the **Options State** command to select the coordinate system used for *subsequent* 2D plots. If the Rectangular option is selected, expressions are plotted in rectangular coordinates (see Section 5.1.1). Initially rectangular coordinates are used unless changed by the ***DERIVE*** initialization file (see Section 2.14).

If the Polar option is selected, expressions are plotted in polar coordinates, and the 2D-plot window's **Plot** command prompts you to enter minimum and maximum angular values for the variable. The initial values are numerical approximations of $-\pi$ and π for radian angle mode and -180 and 180 for degree angle mode. Press **Tab** to move between fields. If you wish to change the initial values, enter radians if the angle mode is Radian and degrees if the angle mode is Degree.

You also have the option to plot a continuous line or discrete points. The default is a continuous plot line. For example, plotting $\theta^2/3$ as a continuous plot line with θ going from $-\pi$ to π produces



A Polar Plot

If Discrete is selected in the Mode field, you may enter in the Points field the number of equally spaced values of the parameter to plot. Note that in polar plot mode the position of the cross is displayed in polar coordinates on the status line. The radial distance of the cross from the origin is labeled by an *r*. The angle of the cross, represented in either radians or degrees depending on the angle mode, is measured counterclockwise from the positive *x*-axis and labeled by the Greek letter θ.

Since polar plots are generated in a standard 2D-plot window, you can zoom, move the cross, and plot any mixture of expressions as described previously.

If a vector of *three* or more expressions is plotted in polar plot mode, each element of the vector is plotted individually. This makes it convenient to plot a family of curves.

Exercise: Make polar plots of the family of curves

```
SIN(θ),  SIN(2θ),  SIN(3θ)
```

etc. until you can predict the appearance of the next plot.

Trace mode is available for polar plots as well as rectangular plots (see Section 5.1.1). When you start tracing a polar plot, the cross goes to the midpoint of the parameter range. The midpoint of the parameter range is calculated from the value in

the Points field when plotted in discrete mode. The midpoint of a continuous plot is calculated from the current value in the Points field.

When tracing parametric or polar plots the message line shows a message of the form

Tracing expression #*n*, at t = x.xxxxxxx

where *n* is the number of the expression being traced and x.xxxxxxx is the parameter value at the box location. For polar plots, the parameter value represents the angle.

For tracing purposes, the angular range of the polar plot is divided into equally spaced points. The number of points is 10 times the number entered in the Points field. When tracing a polar plot, use the direction keys to move the box along the plot as follows:

- **Left arrow** or **Ctrl-S** decreases the parameter value to move the box the distance of one point along the plot.
- **Right arrow** or **Ctrl-D** increases the parameter value to move the box the distance of one point along the plot.
- **Ctrl-Left arrow** or **Ctrl-A** decreases the parameter value to move the box the distance of ten points along the plot.
- **Ctrl-Right arrow** or **Ctrl-F** increases the parameter value to move the box the distance of ten points along the plot.

The **Up arrow** and **Down arrow** keys may be used to trace another rectangular or polar plot line in a 2D-plot window with multiple expressions plotted (see Section 5.1.4). The **PgUp** and **PgDn** keys are disabled in trace mode.

The polar plot line only spans expression values from the minimum parameter to the maximum parameter specified. When tracing a polar plot, the box can be moved off the edge of the plot line in order to visualize the shape of the plot if the parameter range were extended.

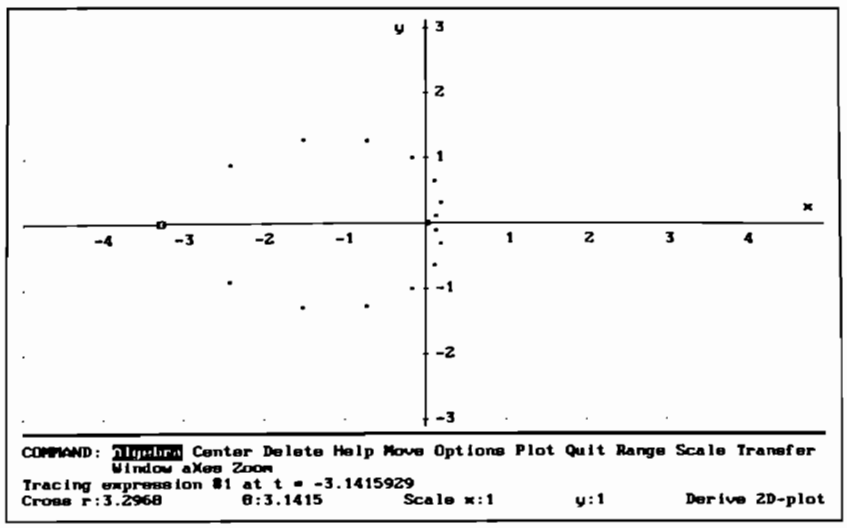
Exercise: Make a polar plot of the following expression

$$\frac{\theta^2}{3}$$

from $-\pi$ to π using 19 discrete points. Turn on trace mode. Press **Ctrl-Left arrow** until the box is positioned over the parameter minimum.

Hint: Look for a *t* value of $-\pi$ on the message line.

The screen should look similar to:



Tracing a Plot of Discrete Points

Now, use **Ctrl-Right arrow** to trace to the parameter maximum. Use the left and right arrow keys to trace in between the discrete points. Finally, turn off trace mode.

5.1.7 Parametric Plots

A **parametric plot** results from plotting a vector of two univariate expressions dependent on the same variable. In rectangular plot mode the value of the first expression is the x-coordinate, and the value of the second expression is the y-coordinate. In polar plot mode the value of the first expression is the radial distance from the origin. The value of the second expression is the angle, represented in either radians or degrees depending on the angle mode (see Section 6.3), and measured counterclockwise from the positive x-axis.

Use the following procedure to make a parametric plot:

- First, in an algebra window, enter and highlight a pair of expressions dependent on the same parameter, separated by a comma and enclosed in square brackets.
- Next, use the algebra window's **Plot** command to open and switch to a 2D-plot window.

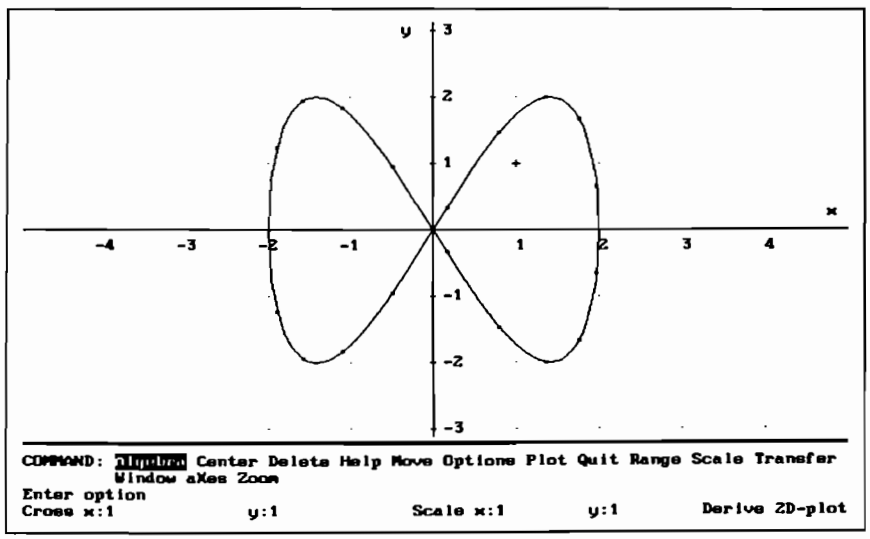
- Finally, issue the 2D-plot window's **Plot** command and enter minimum and maximum values for the parameter. Press **Tab** to move between fields. The initial values are numerical approximations of $-\pi$ and π for radian angle mode and -180 and 180 for degree angle mode. These values are often appropriate when plotting pairs of sinusoids.

For parametric and polar plots, the **Plot** command gives you the option to plot a continuous plot line or discrete points. If discrete points is selected, also enter the number of equally spaced values that you want the parameter to assume. Discrete parametric plots give you a sense of how the parameter varies along the curve.

Exercise: Make a continuous parametric plot in rectangular mode of the expressions

```
[2·COS(t), 2·SIN(2·t)]
```

from $-\pi$ to π. Then replot the expression using 20 discrete points so that the discrete points are superimposed on the continuous curve. The resulting plot should look similar to:



A Parametric Plot

To plot multiple parametric expression pairs with a single **Plot** command, enter and highlight a vector of expression pairs (see Section 5.1.4). For each expression pair, you are prompted to enter the minimum and maximum values for the parameter. To avoid this and use the same values for all the remaining plots, press **Ctrl-Enter** or **Ctrl-J** instead of **Enter** in response to the prompt.

When making a continuous parametric plot, the parameter interval is recursively subdivided until the curve can be accurately approximated by a straight line. This may take an unacceptably long time if subdividing the curve is outside the window or in the neighborhood of a singularity. If this occurs, use the **Delete** command to delete the continuous plot and replot the parametric expressions using discrete points.

Exercise: Make a discrete parametric plot in rectangular mode of the pair of expressions

```
[t, 1/t]
```

from $-\pi$ to π using 20 discrete points. Then attempt to make a continuous parametric plot of the expression. Determine why ***DERIVE*** is unable to complete the plot. Press **Esc** to abort the plot.

Since parametric plots are done in a standard 2D-plot window, you can zoom, move the cross, and plot any mixture of expressions as described previously. Tracing a parametric plot works in the same way as tracing a polar plot (see Section 5.1.6).

For a collection of parametric expressions that are interesting to plot, load the file PLOTPARA.MTH using a **Transfer Merge** command (see Section 2.9).

Functions for plotting parametrically defined space curves such as helixes are defined in GRAPHICS.MTH (see Section 9.19).

5.1.8 Data Matrix Plots

A data point is parametrically represented as a vector of two numbers. For example, [2, 3] is the point whose x-coordinate is 2 and whose y-coordinate is 3.

A data point plots as a single point. Since there is no parameter, the **Plot** command does not prompt for minimum and maximum values when plotting data points. Note that in polar plot mode data points are plotted in polar coordinates.

A data matrix is a vector of data points. When a data matrix is plotted, each point is plotted individually, making it convenient to plot a collection of data points. For example, the data matrix

```
[[1, 1], [-2, 2], [0, 1], [2, 0]]
```

plots as 4 data points.

To generate a data matrix for an expression u involving x as x goes from a to b in steps of size s, enter a VECTOR command of the form

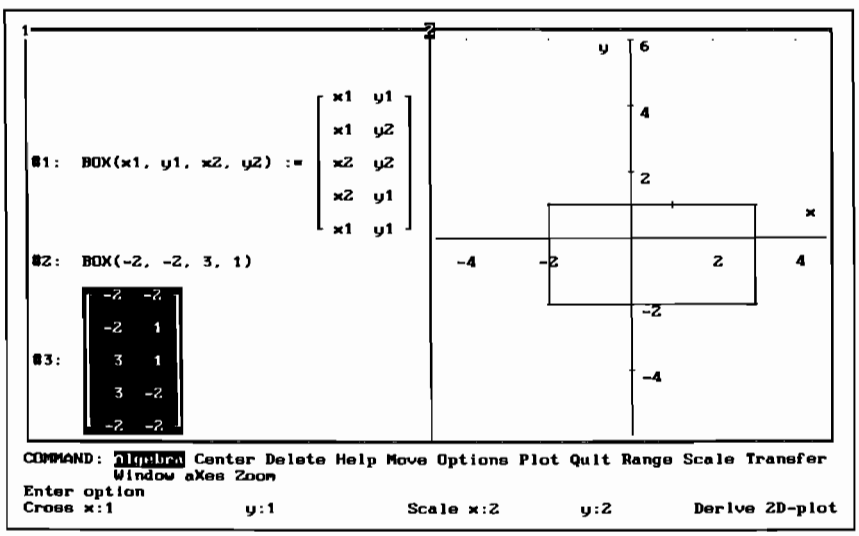
VECTOR($[x, u]$, x, a, b, s)

If, instead of both being numbers, any of the coordinate pairs depends upon the same one variable, then that pair is plotted parametrically after prompting for the parameter range.

Use the **Options Color Auto** command to plot all the data points in the same color. Use the Mode field of the **Options State** command to connect the points of a data matrix with line segments. Use the Size field of the **Options State** command to determine the size (Large or Small) of the data points. A large data point displays as a solid 3-pixel by 3-pixel rectangle while a small data point displays as a 1 pixel rectangle.

Exercise: Define the function BOX(x_1, y_1, x_2, y_2) which simplifies to an expression that plots as a box with opposite corners at (x_1, y_1) and (x_2, y_2) when vectors of data points are connected by line segments. Use BOX to draw a box with corners at the points (–2, –2) and (3, 1).

If you vertically split the ***DERIVE*** screen (see Section 2.13) and zoom out in both directions, the resulting plot should look similar to:



A Data Matrix Plot

Note that BOX(x_1, y_1, x_2, y_2) starts and finishes with the point [x_1, y_1]. This is required to close the box when ***DERIVE*** is in connected plot point mode.

5.1.9 Implicit Plots

Functions implicitly defined by an equation may be plotted using a relatively efficient algorithm of linear interpolation upon triangles. As described in Section 5.1, the form of an implicitly defined equation is $u = v$, where u and v are expressions of at most two variables. Examples of such equations are

$$x^2 + y^2 = 4$$

$$x^3 = y^2 - 5$$

$$x = 3$$

$$r^2 + \cos(\theta) = 0$$

To produce an implicit plot in rectangular plot mode, the horizontal and vertical axes are assigned to variables in the equation. Usually, the primary variable is assigned to the horizontal axis, the secondary variable to the vertical axis (see Section 4.3). However, if there is only one variable, x, as in the third equation above, x will be assigned to the horizontal axis; otherwise the variable will be assigned to the vertical axis. For implicit plotting in rectangular mode we recommend using x for the horizontal and y for the vertical coordinate.

To produce an implicit plot in polar plot mode, the angle and the distance from the origin are assigned to variables in the equation. Usually, the primary variable is assigned to the angle, the secondary variable to the distance from the origin. However, if one of the variables in the equation is a Greek letter, it is assigned to the angle. For implicit plotting in polar mode we recommend using r for the distance from the origin and θ for the angle.

Exercise: Make a rectangular mode plot of the equation

$$x^2 + y^2 = 4$$

Note that it is a circle. Use the **Manage Substitute** command to substitute $r\cos(\theta)$ for x and $r\sin(\theta)$ for y and produce the following equation

$$(r\cdot\cos(\theta))^2 + (r\cdot\sin(\theta))^2 = 4$$

Plot this equation in polar mode. Note that these are equivalent ways to plot the same circle.

The accuracy of an implicit plot may be adjusted by using the **Options Accuracy** command. A plot accuracy of 9 will give the most accurate plot with the slowest

plotting speed, while a plot accuracy of 0 gives the least accurate plot with the fastest plotting speed. Note that the accuracy of an implicit plot in text mode is much less than the same plot in graphics mode.

The implicit plotting algorithm starts in the top left corner of the 2D-plot window, plotting points in a top to bottom, left to right fashion. As with explicit plots, you may zoom, scale, center, or stop an implicit plot before it is complete. Unlike most explicit plots, implicit plots may not be automatically scaled; nor may an implicit plot line be traced.

Explicit plots are faster and somewhat more accurate than implicit plots. Before plotting, when possible, ***DERIVE*** automatically converts a function implicitly defined by an equation into a function explicitly defined by an equation of the form $y = u$ where y is a variable and u is a univariate expression in another variable. For example, the linear equation

$$2x + 3y = 6$$

is transformed into the equation

$$y = -2/3x + 2$$

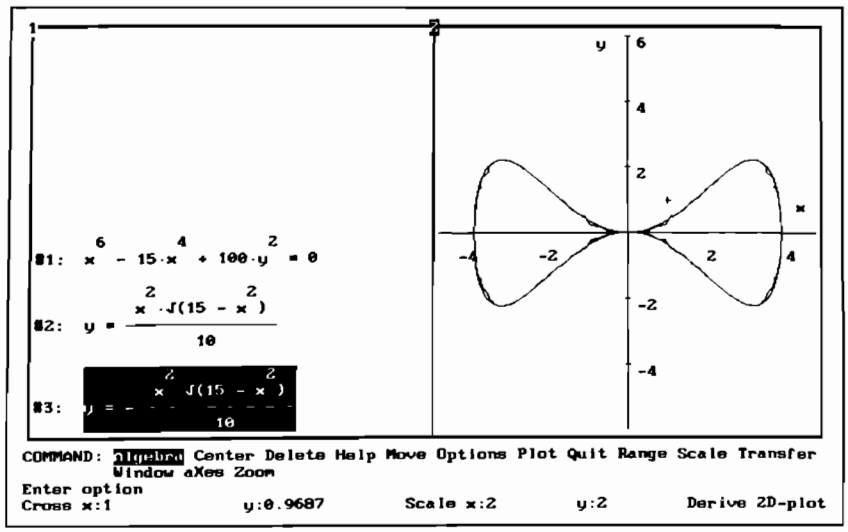
and then plotted using the more expeditious explicit plotting algorithm.

A function implicitly defined by an equation may be solved for one of its variables in terms of the other by using the algebra window's **soLve** command to generate explicit solutions. If explicit solutions of the equation are achievable, explicit plots of these solutions may be superimposed in a 2D-plot window.

Exercise: Plot the following implicitly defined function in rectangular mode

$$x^6 - 15 \cdot x^4 + 100 \cdot y^2 = 0$$

in a 2D-plot window beside the algebra window. Zoom out in both directions so that all of the plot is visible. Now, switch back to the algebra window and solve the equation for y. Plot the two explicit solutions in the 2D-plot window. Note the areas where the implicit plot fills in more of the curve than the two explicit plots. The screen should look similar to:


#1: x^6 - 15·x^4 + 100·y^2 = 0
#2: y = x^2·√(15 - x^2) / 10
#3: y = -x^2·√(15 - x^2) / 10
COMMAND: Algebra Center Delete Help Move Options Plot Quit Range Scale Transfer Window aXes Zoom
Enter option
Cross x:1 y:0.9687 Scale x:2 y:2 Derive 2D-plot

A Bow-Tie Plot

An alternate method for generating explicit solutions for implicitly defined equations is to substitute polar coordinates for the variables and solve for the distance from the origin in terms of the angle. For example,

$$x^5 + x^4 = y^5 + y^4$$

cannot be solved exactly for either x or y. However, substituting $r\cos(\theta)$ for x and $r\sin(\theta)$ for y, then solving for r gives the two explicit solutions

$$r = 0$$

$$r = \frac{1 - 2\cos(\theta)^2}{\cos(\theta)^5 - \sin(\theta)^5}$$

These explicit solutions may then be plotted in polar plot mode in the 2D-plot window.

Implicit 2D plots of a family of plot lines can also be used to make contour plots of functions of two variables. Simplifying and then plotting an expression of the form

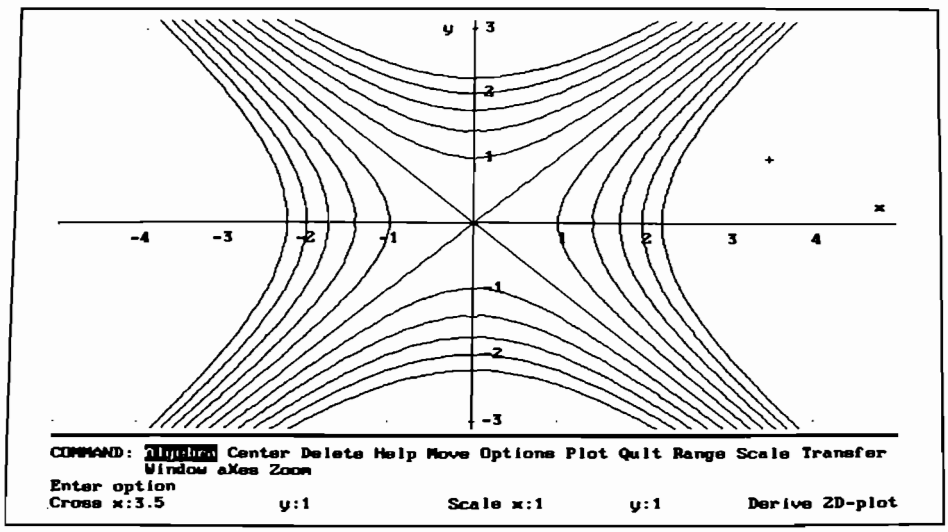
VECTOR ($z = u, z, m, n, s$)

produces a contour plot of the function $z = u$ where u is a function of x and y as z varies from m to n in steps of size s. The **Calculus Vector** command is an easy way to enter such expressions (see Section 8.2).

Exercise: Make a contour plot of the function

$$z = x^2 - y^2$$

as z varies from −5 to 5 in steps of 1. The screen should look similar to:



A Contour Plot

5.2 3D-plot Window

DERIVE's 3D-plot window is used for plotting the following types of expressions and equations generated in an algebra window:

- expressions involving two variables (i.e. bivariate expressions); and
- equations of the form $z = u$, where z is a variable and u is a bivariate expression in two other variables.

An expression containing two variables defines a surface imbedded in a three dimensional space. Evaluating such a bivariate expression over a uniformly spaced grid of values for the two variables yields an array of points on the surface. Connecting adjacent points by straight line segments produces a **wire frame grid** that approximates the shape of the surface. Projecting and plotting this wire frame grid on a two dimensional computer screen allows you to visualize the three dimensional surface.

Section 5.2.1 describes how to make a 3-dimensional wire-frame plot of a bivariate expression. Section 5.2.2 describes how to optimally frame bivariate plots. Section 5.2.3 describes how to rotate a 3-dimensional plot by modifying the eye coordinates and focal point. Section 5.2.4 describes how 3-dimensional plots can be manipulated using the **AcroSpin** software.

5.2.1 Plotting Bivariate Expressions

Use the following procedure to plot a bivariate expression, subexpression, or equation whose right side is a bivariate expression:

- In an algebra window, highlight the expression, subexpression, or equation to be plotted (see Section 3.3).
- Issue the algebra window's **Plot** command to open and switch to a 3D-plot window overlaying the algebra window. Alternatively, you may want to split the window and designate one of the windows a 3D-plot window (see Section 2.13). In either case, the phrase

  Derive 3D-plot

  is displayed at the right end of the status line, indicating that a 3D-plot window is currently active.

- Issue the 3D-plot window's **Plot** command to actually plot the highlighted expression. The message line successively announces that it is computing expression values, computing the perspective projection, and plotting the expression. If you wish to stop the plotting before completion, press the **Esc** key.
- On PC compatible computers, to send a graphics image of a *DERIVE* screen including a 3D-plot window to a printer, press **Shift-F10**. To send just the plot *without* the menus and status line at the bottom of the screen, press **Shift-F9**. Before printing you must select the appropriate printer type using the **Transfer Print Options** command (see Sections 2.10 and 2.11).

It is also possible to plot a constant or a univariate expression in a 3D-plot window. After highlighting such an expression, use the **Window Designate** command and select the 3D-plot window type (see Section 2.13) instead of using the algebra window's **Plot** command.

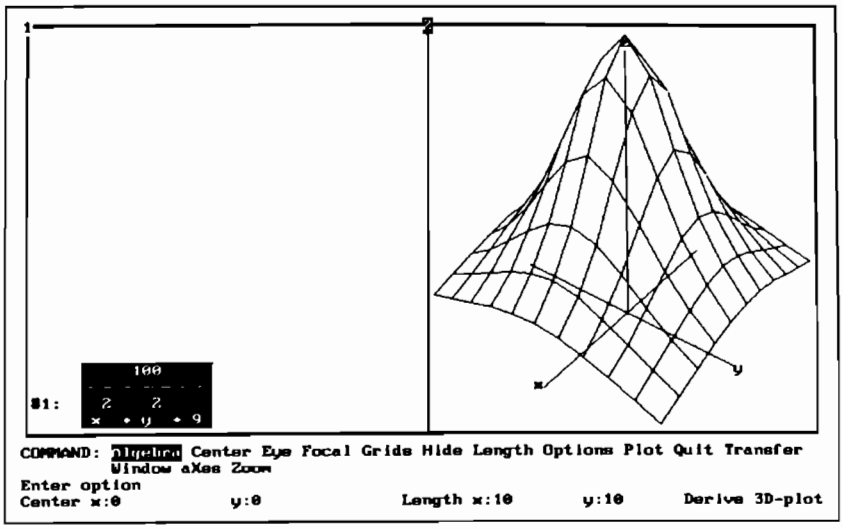
Exercise: Plot the expression

$$\frac{100}{x^2 + y^2 + 9}$$

or the equation

$$z = \frac{100}{x^2 + y^2 + 9}$$

If you vertically split the *DERIVE* screen, the plot should look similar to:



A Wire-frame Plot with Hidden Lines Removed

Regardless of what the two independent variables in your expression are named, they are denoted by x and y on the plot and on the status line. The expression value is denoted by z. Initially:

- The positive x direction comes toward you and down to the left.
- The positive y direction comes toward you and down to the right.
- The positive z direction comes toward you and up.

In 3D plots of a bivariate expression, if neither variable is x, the expression's primary variable (see Section 4.3) is denoted by x and its other variable is denoted by y.

Initially, the 3D-plot window shows the three axes with their positive directions labeled by x, y, and z. This is helpful in orienting yourself but sometimes clutters the image. Use the **aXes** command to control the display of the axes. If you do not want axes displayed on *subsequent* 3D-plots, select No in response to the **aXes** command.

Exercise: Replot the expression in the previous exercise with the axes removed.

For 3D plots, the expression is evaluated on a rectangular grid of points that is uniformly spaced in x and uniformly spaced in y. The status line displays the coordinates of the center of the rectangle and the lengths of the sides of the rectangle.

The values of the expressions are interpolated by straight-line segments along the lines of constant x and along the lines of constant y. This gives a wire-frame image. Only real finite values are plotted and connected to their neighbors.

The default is to *not* plot those segments that would be hidden if the surface were opaque. Use the **Hide** command to control the plotting of these "hidden" lines. If you want hidden lines plotted on *subsequent* 3D-plots, select No in response to the **Hide** command.

If ***DERIVE*** is *not* plotting hidden lines, it plots the top of the surface and then the bottom of the surface. After it is clear that the rest of the bottom surface is hidden, press **Esc** if you are in a hurry to proceed with other commands. Removing hidden lines is slower than plotting them because each plot point must be checked for visibility.

Exercise: Replot the expression in the previous exercise with the hidden lines plotted. Compare the time required to display the plot with and without hidden line removal.

You can increase or decrease the number of grid lines in either the x or y direction or both. Increasing the grid lines allows more surface detail to be resolved, until a point is reached where the grid lines start running together. This point of diminishing returns depends on the expression, the resolution of your graphics mode, and the size of your plot window. It is rarely worth going beyond about 40 by 40 grid lines.

Use the **Grids** command to change the number of grid lines. This command prompts you to enter the desired number of grid panels in the x and y directions. Note that the number of grid panels is one less than the number of grid lines for each direction.

The computation time and storage requirements increase with the number of grid panels. A good strategy is to use a rather coarse grid until you are satisfied with the orientation and perspective of the plot.

If the warning message Memory Full is displayed while computing a 3D plot, you must reduce the number of grid panels to complete the plot.

Use the **Options Color Plot** command to change the color of the axes and the top and bottom surfaces of the *next* expression that is plotted. This command displays the integers 0 through 15 on the message line, each number displayed in its respective color, and prompts you to enter a color number. You can enter any integer from 0 to

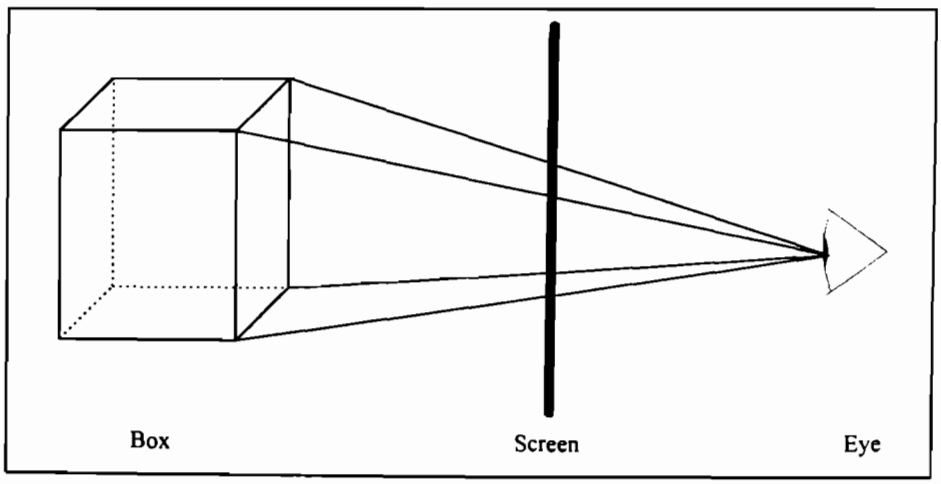
15, however the actual number of colors available depends on your computer's graphics hardware and the current display mode (see Section 2.12).

For a collection of bivariate expressions that are interesting to plot in 3D, load the file PLOT3D.MTH using a **Transfer Merge** command (see Section 2.9).

Functions for projecting parametrically defined surfaces such as spheres, tori, and cones so that they can be plotted in a 2D-plot window are defined in GRAPHICS.MTH (see Section 9.19).

5.2.2 Framing 3D Plots

Imagine that the surface of an expression being plotted is enclosed in a transparent box just spanning the maximum and minimum computed values of x, y, and z. Your monitor screen is between the box and an eye fixed somewhere in front of the screen as illustrated by the following diagram:



Each point on the wire frame projects onto the screen along the line joining that point to the eye.

For maximum resolution it is usually desirable to magnify the surface as much as possible while keeping it entirely within the 3D-plot window. Therefore, ***DERIVE*** automatically magnifies the image horizontally so the surface just fits the full width of the 3D-plot window. Also, if using the optional automatic vertical centering and scaling, ***DERIVE*** automatically tries to magnify and shift the image vertically so the

surface just fits the height of the 3D-plot window. Naturally this is impossible if the surface projects as a horizontal line, in which case ***DERIVE*** generates an error beep.

You may want to defeat automatic vertical centering and/or scaling to impose a particular scaling in all three directions. For example, you may want to impose equal scaling in all three directions. Alternatively, you may want to magnify certain details at the expense of having portions clipped at the top or bottom edge of the plot window.

Use the **Center** command to display and/or set the coordinates of the center of the transparent box and to control whether or not automatic vertical centering is done. The initial center is at $x=0$ and $y=0$, with automatic vertical centering turned on. Changing the vertical center (i.e. z coordinate) will also turn off automatic vertical centering.

Use the **Length** command to display and/or set the lengths of the sides of the transparent box and to control whether or not automatic vertical scaling is done. The initial box lengths are 10 by 10 with automatic vertical scaling turned on. Changing the vertical length (i.e. z interval length) also turns off automatic vertical scaling.

Use the **Zoom** command to change the lengths of the sides of the transparent box. Zooming in magnifies surface detail by displaying a smaller portion of the surface and zooming out shrinks surface detail by displaying a larger portion of the surface. You can zoom in the x direction, the y direction, or both directions at once. When you zoom in or out, the length is changed to the next closest number in the sequence: ... 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20 ... .

The function keys provide an alternative to using the **Zoom** command. The following function keys apply to 3D plots as well as 2D plots. To magnify:

- **F7** zooms the y direction in.
- **Shift-F7** zooms the x direction in.
- **F9** zooms both directions in.

The following function keys shrink the plot:

- **F8** zooms the y direction out.
- **Shift-F8** zooms the x direction out.
- **F10** zooms both directions out.

Exercise: Plot the expression

$$\frac{100}{9 + x^2 + (y - 3)^2} + \frac{100}{9 + x^2 + (y + 3)^2}$$

using various settings for the center point, the number of grids, the box lengths, and the scale.

The framing challenge for three dimensions is more difficult than for two dimensions because of the lower effective resolution and the extra dimension to search. If a few experiments do not reveal appropriate ranges for the two variables in your expression or equation, then use a 2D-plot window to plot cross-sections obtained by substituting various numeric values for one or the other of the two variables. The ten-fold greater resolution of 2D plots often more quickly reveals appropriate ranges for viewing the two variables in a 3D-plot window. If you zoom in on a 2D plot until the features of interest are at least 1/*n*th of the window width, they have a chance of being resolved in a 3D-plot window of that size using *n* grid lines for that coordinate.

5.2.3 Rotating 3D Plots

After you have optimally framed a 3D plot, it might be helpful to circumnavigate the transparent box by changing the horizontal eye coordinates in a way that keeps the distance to the nearest point on the surface about the same.

Use the **Eye** command to set the coordinates of the viewer's eye relative to the transparent box and to control whether or not automatic vertical positioning is used for the eye. When a **Center**, **Length**, or **Zoom** command is issued, the coordinates of the eye are automatically adjusted to maintain the eye in the same position relative to the center and size of the transparent box.

Change the *z* coordinate if you want to view the surface from a steeper or shallower angle, perhaps from underneath. Change the *x* and *y* coordinates in concert in order to move closer to or farther from the center along the current line of sight from the focus. Moving the eye relatively closer to the box exaggerates the perspective distortion, and moving the eye farther away lessens it. Note that you will get an error beep if you try to position the eye directly above the center of the box.

When changing the eye position it is important to remember that the image is scaled horizontally to just fit the 3D-plot window width and usually scaled vertically to just fit the window height. Consequently, the transparent box and the surface being plotted will not shrink towards the box center as you back away (zoom out). If you want that effect, just move away from your computer screen or make a smaller

plot window! Conversely, the grid lines will not disappear off the edges of the screen as you approach the surface (zoom in). If you want that effect, zoom in while reducing the number of grid lines. The 3D-plot window is designed to reveal maximum-resolution qualitative information about bivariate expressions rather than simulating flight over a landscape or architectural collections of geometric shapes.

Use the **Focal** command to set the coordinates of the focal point of your field of vision and to control whether or not automatic vertical positioning is used for the focal point. Making the x and y components of the focal point disparate, such as positive for one and negative for the other, makes the image appear as though seen at the edge of your field of vision. When a **Center** command is issued, the coordinates of the focal point are automatically set to the center of the transparent box. It is rarely advantageous to alter the focal point coordinates.

Exercise: Using the same plot as in the previous exercise, rotate your viewpoint by adjusting the eye and focal coordinates.

5.2.4 Animation with AcroSpin

AcroSpin is a program for PC compatible computers that allows you to rotate, translate (reposition the image on the screen), and scale (enlarge or shrink) 3-dimensional wire frame objects and point clouds. **AcroSpin** performs these transformations so quickly that the objects appear to move on your screen, just as the real objects would move in space if you performed the same transformations on them.

AcroSpin reads a description of the wire frame object or point cloud from a data file on disk. Once **AcroSpin** has read the data file, you control the motion of the object from your keyboard. You can scale the object by a factor of about 8000, rotate around any of the three coordinate axes, and translate the object left and right or up and down. Using the keyboard, you can animate multiple objects and turn the display of individual objects on and off.

If you have the **AcroSpin** software, you may use **AcroSpin** to manipulate a 3D plot created in ***DERIVE***. **AcroSpin** is a product of Acrobits and may be purchased from MathWare, P.O. Box 3025, Urbana, IL 61801, USA, phone (800) 255-2468, fax (217) 384-7043.

Use the following procedure to invoke **AcroSpin** from ***DERIVE***:

- In an algebra window, highlight a bivariate expression and switch to a 3D-plot window as described in Section 5.2.1.
- Issue the **Transfer Acrospin Save** command. The plot data points will be evaluated and readied for saving in a data file.

- You will be prompted to enter the drive, directory, name, and extension of the data file. If you do not specify a drive and/or directory, the file is saved on the current DOS drive and directory. If you do not specify an extension, the file name extension ACD is used.
- Once the data points are saved in a file you will be asked whether or not you want to run **AcroSpin**. Type a Y at the prompt to run **AcroSpin** using the set of data points you just saved from the ***DERIVE*** 3D-plot window. Type N at the prompt if you do not want to run **AcroSpin** at this time.

Note that the following 3D-plot window settings affect the **AcroSpin** data file: the top surface color; the number of grid lines; the center point; and the length.

AcroSpin uses a three dimensional coordinate system. The origin of the coordinate system is at the center of your screen. The *x*-axis runs horizontally from left to right, the *y*-axis runs vertically from bottom to top, and the *z*-axis comes out from the screen toward you. Since the orientation of the axes in **AcroSpin** differs from that of ***DERIVE***, the plot image initially projected by **AcroSpin** will be rotated so that you are looking up from the bottom of the plot. Hold down one of the arrow keys to rotate the plot until it is recognizable.

More detailed information about **AcroSpin** may be found in the **AcroSpin** user manual. On-line help may be obtained by pressing **h** once inside **AcroSpin**.

To quit **AcroSpin** and return to ***DERIVE***, press the **Esc** key. Once in ***DERIVE***, you can replot your expression if so desired.

Exercise: Plot the expression

$$\frac{200}{x^2 + y^2 + 9}$$

Now, issue the **Transfer Acrospin Save** command. Name the file PAPERHAT. Type Y when prompted to run **AcroSpin**. When **AcroSpin** comes up, press and hold down the **Right arrow** key to rotate the plot image until the shape of the hat becomes visible. Press **h** to read the **AcroSpin** on-line help. Experiment with other **AcroSpin** commands. Finally, press **Esc** to return to ***DERIVE***.

You can superimpose several 3D plots in **AcroSpin** by saving the data points for the plots in the same data file. Use the procedure described previously to save the data points for the first plot. Do not run **AcroSpin** yet. To save the data points for the next plot, switch back to the algebra window, highlight another expression, switch back to the 3D-plot window, and use the same procedure and the *same* file name. Type **N** when asked if you want to *overwrite* the existing file. Type Y when asked if

you wish to *append* to the existing file. If this is the last plot you wish to superimpose, then select Y to run **AcroSpin**. Otherwise, continue appending data points from all the desired plots to the same data file. For easier visual identification in **AcroSpin**, change the top surface color of each successive plot before appending its data points to the data file.

Exercise: Append data points for the expression $z=12$ to the PAPERHAT.ACD **AcroSpin** data file of the previous exercise. Run **AcroSpin** with this new data file.

> **Hint**: After you have authored the equation $z=12$, press **F1** or **F2** as appropriate to switch back to the 3D-plot window (i.e., do not use the **Plot** command from the algebra window since that will switch to a 2D-plot window). Remember to change the plot color before appending the data points.

Unlike ***DERIVE***, **AcroSpin** does not automatically scale the z coordinate. If the plot image you see in **AcroSpin** looks like a flat grid and rotating it does nothing to make it look similar to the ***DERIVE*** plot, you will need to scale the original equation yourself in the algebra window.

Exercise: Plot the expression

$$\frac{1}{x^2 + y^2 + 9}$$

Issue the **Transfer Acrospin Save** command and name the file GRID. Type Y when prompted to run **AcroSpin**. Try to rotate this image so the shape of the hat becomes visible. Determine why it looks like a flat grid instead of the hat that was plotted in the ***DERIVE*** 3D-plot window.

> **Hint**: Look at the interval lengths displayed in the 3D-plot window by the **Length** command. What happens to the plot if you modify the z interval length to be the same as the x and y lengths?

If you have previously saved an **AcroSpin** data file, **AcroSpin** may be run by issuing the **Transfer Acrospin Run** command from the 3D-plot window and choosing the data file you wish to use. You are prompted to enter the drive, directory, name, and extension of the data file. If you do not specify a drive and/or directory, the file is read from the current DOS drive and directory, in the \DERIVE directory on the current drive, or in the directory from which ***DERIVE*** was invoked. If you do not specify an extension, the file name extension ACD is used.

Chapter 6. Functions and Constants

This chapter describes ***DERIVE***'s pre-defined mathematical functions and constants. Chapter 9 describes the numerous functions defined in the utility files distributed with ***DERIVE***. Section 4.12 and Chapter 10 describe how to define your own functions in terms of ***DERIVE***'s pre-defined operators, functions, and constants.

Numerous automatic and some optional transformations are used to simplify expressions involving the pre-defined functions and constants. To see a demonstration of these simplifications, load the files TRIG.DMO and FUNCTION.DMO using a **Transfer Demo** command (see Section 2.9).

Note that these transformations are not applied to an expression until you issue a **Simplify**, **Expand**, or **Factor** command (see Sections 4.2, 4.4, and 4.6). Merely entering an expression using an **Author** command or building an expression using a **Build** command is not sufficient to force simplification, unless **Ctrl-Enter** or **Ctrl-J** instead of **Enter** is used to issue these commands (see Sections 3.2 and 3.5).

Some transformations for simplifying the pre-defined functions are valid only for a restricted domain of values. For example, the transformation

$$\frac{\text{LN}(x^2)}{2} \rightarrow \text{LN}(x)$$

is invalid if x is negative.

When simplifying an expression, ***DERIVE*** does not exploit a transformation unless it can determine that the transformation is valid. This determination depends on the domain of the variables in the expression. Use the **Declare Variable Domain** command to restrict the domain of variables so that more transformations can be applied (see Section 4.10).

Given numeric arguments, the value of most pre-defined functions is usually an irrational number which therefore cannot be computed exactly. For example, the square root of most rational numbers is irrational. So in exact mode, only certain special numeric arguments simplify to rational numbers. For example, in exact mode $\sqrt{(4/9)}$ simplifies to 2/3, but $\sqrt{(8/9)}$ simplifies to $(2\sqrt{2})/3$. You can use the **approX** command to temporarily change the precision to obtain an approximate result.

Given a complex argument in rectangular form (see Section 4.5), most pre-defined functions produce a simplified result that is also in rectangular form. For example,

$$\sqrt{(3 - 4 \cdot i)}$$

simplifies to $2 - i$.

6.1 Exponential Functions

- SQRT(z) is the square root of the expression z. SQRT(z) is equivalent to $z^{1/2}$. On PC compatible computers SQRT displays as $\sqrt{}$ and can be entered by pressing **Alt-Q**. If z is a complex expression, then

$$-\frac{\pi}{2} < \text{PHASE}(\sqrt{z}) \leq \frac{\pi}{2}$$

  where PHASE is the phase angle measured in radians (see Section 6.8). The rules used to simplify expressions involving square roots take into account the current branch selection (see Section 4.5) and the declared domain of variables (see Section 4.10).

- #e is the base of the natural logarithms (2.71828...). On PC compatible computers #e displays as ê and can be entered by pressing **Alt-E**. You are free to use the unembellished variable name e to represent energy, eccentricity, etc.

- EXP(z) is the exponential of the expression z. EXP(z) simplifies to $\hat{e}^z$.

  The transformation

$$\hat{e}^z \cdot \hat{e}^w \leftrightarrow \hat{e}^{z+w}$$

is valid if z and w are real or complex expressions. The transformation

$$(\hat{e}^z)^k \leftrightarrow \hat{e}^{k \cdot z}$$

is valid if k is an integer, or if k is a fraction and z is a real or complex expression with

$$-\pi < \text{IM}(z) \leq \pi$$

Use the **Manage Exponential** command to control the direction these exponential transformations are applied in *subsequent* simplifications. This command displays the prompt:

MANAGE EXPONENTIAL: Direction: Auto **Collect** Expand

Select Collect to have both transformations applied toward the right. Select Expand to have both transformations applied toward the left. Select Auto to have the first transformation applied toward the right and the second applied toward the left. Since it usually gives the most compact and attractive form, the initial direction is Collect.

When a new direction is selected, the **Manage Exponential** command generates an expression of the form

Exponential := *direction*

where *direction* is Auto, Collect, or Expand. The exponential transform direction can also be changed by entering on the author line an expression of the above form using upper and lower case exactly as shown.

6.2 Logarithmic Functions

- LN(z) is the principal natural (Napierian) logarithm of the expression z. For example, LN(ê^3) simplifies to 3. If z is a complex expression, then

$$-\pi < \text{IM}(\text{LN}(z)) \leq \pi$$

where IM(z) simplifies to the imaginary part of z (see Section 6.8).

- LOG(z, w) is the logarithm of z to the base w. For example, LOG(8, 2) simplifies to 3. LOG(z, w) simplifies to LN(z)/LN(w). LOG(z) simplifies to LN(z).

The transformation

$$\text{LN}(x) + \text{LN}(z) \quad \leftrightarrow \quad \text{LN}(x \cdot z)$$

is valid if z is real or complex and x is nonnegative. The transformation

$$k \cdot \text{LN}(x) \quad \leftrightarrow \quad \text{LN}(x^k)$$

is valid if k is a rational number, and x is nonnegative or $-1 < k \leq 1$.

Use the **Manage Logarithm** command to control the direction these logarithmic transformations are applied in *subsequent* simplifications. It displays the prompt:

MANAGE LOGARITHM: Direction: **Auto** Collect Expand

Select Collect to have both transformations applied toward the right. Select Expand to have both transformations applied toward the left. Select Auto to have the first transformation applied toward the right and the second applied toward the left. Since it usually gives the most compact and attractive form, the initial direction is Auto.

When a new direction is selected, the **Manage Logarithm** command generates an expression of the form

Logarithm := *direction*

where *direction* is Auto, Collect, or Expand. The logarithm transform direction can also be changed by entering on the author line an expression of the above form using upper and lower case exactly as shown.

Exercise: See which of the three **Manage Logarithm** command settings you prefer when simplifying

$$2 \cdot \text{LN}\left[\frac{4 \cdot x}{3}\right] - \text{LN}\left[\frac{20}{9}\right]$$

6.3 Trigonometric Functions

The trigonometric (circular) functions take an angle measured in the angular unit (degree or radian) specified by the Angle field of the **Manage Trigonometry** command (see below). The following discussion assumes the angular unit is a radian.

- pi is the area of a unit circle (3.14159...). pi displays as π and can be entered by pressing **Alt-P** on PC compatible computers. In exact mode pi remains symbolic; otherwise it simplifies to an approximate value accurate to the current precision.
- deg is the number of radians per degree (i.e. $\pi/180$). On PC compatible computers deg displays as °. Multiply an angle measured in degrees by deg to convert it to radians. For example, 60·deg simplifies to $\pi/3$ radians.
- SIN(z) is the sine of the angle z.
- COS(z) is the cosine of the angle z.
- TAN(z) is the tangent of the angle z. TAN(z) is equal to SIN(z)/COS(z).
- COT(z) is the cotangent of the angle z. COT(z) is equal to COS(z)/SIN(z).
- SEC(z) is the secant of the angle z. SEC(z) simplifies to 1/COS(z).
- CSC(z) is the cosecant of the angle z. CSC(z) simplifies to 1/SIN(z).

Use the **Manage Trigonometry** command to control the application of the trigonometric transformations during *subsequent* simplifications. This command displays the prompt:

MANAGE TRIG: Direction: **Auto** Collect Expand Toward: **Auto** Sines Cosines
Angle: Degree **Radian**

Use the Direction field to control angle-sum and multiple-angle transformations.

When a new direction is selected, the **Manage Trigonometry** command generates an expression of the form

Trigonometry := *direction*

where *direction* is Auto, Collect, or Expand. The trigonometry transform direction can also be changed by entering on the author line an expression of the above form using upper and lower case exactly as shown.

Select the Expand direction to apply the transformations

$$\mathrm{SIN}(z + w) \rightarrow \mathrm{SIN}(z)\cdot\mathrm{COS}(w) + \mathrm{COS}(z)\cdot\mathrm{SIN}(w)$$

$$\mathrm{COS}(z + w) \rightarrow \mathrm{COS}(z)\cdot\mathrm{COS}(w) - \mathrm{SIN}(z)\cdot\mathrm{SIN}(w)$$

$$\mathrm{SIN}(m\cdot z) \rightarrow 2\cdot\mathrm{COS}((m - 1)\cdot z)\cdot\mathrm{SIN}(z) + \mathrm{SIN}((m - 2)\cdot z)$$

$$\mathrm{COS}(m\cdot z) \rightarrow 2\cdot\mathrm{COS}((m - 1)\cdot z)\cdot\mathrm{COS}(z) + \mathrm{COS}((m - 2)\cdot z)$$

where m is a positive integer greater than 1.

Also select the Expand direction to transform trigonometric functions with arguments of the form π/n, where n is a power of 2 times 1, 3, 5, or 15, into algebraic numbers using radicals. For example, SIN(π/24) expands to

$$\sqrt{\left[-\frac{\sqrt{2}}{8} - \frac{\sqrt{6}}{8} + \frac{1}{2}\right]}$$

Exercise: For m = 0 to 14, find the radical representation for SIN($m\pi$/15) and TAN($m\pi$/15).

Select the Collect direction to apply the converse transformations

$$\mathrm{SIN}(z)\cdot\mathrm{SIN}(w) \rightarrow \frac{\mathrm{COS}(z - w) - \mathrm{COS}(z + w)}{2}$$

$$\mathrm{COS}(z)\cdot\mathrm{COS}(w) \rightarrow \frac{\mathrm{COS}(z - w) + \mathrm{COS}(z + w)}{2}$$

$$\mathrm{SIN}(z)\cdot\mathrm{COS}(w) \rightarrow \frac{\mathrm{SIN}(z - w) + \mathrm{SIN}(z + w)}{2}$$

$$\mathrm{SIN}(z)^m \rightarrow \frac{1 - \mathrm{COS}(2\cdot z)}{2}\,\mathrm{SIN}(z)^{m - 2}$$

$$\mathrm{COS}(z)^m \rightarrow \frac{1 + \mathrm{COS}(2\cdot z)}{2}\,\mathrm{COS}(z)^{m - 2}$$

$$a\cdot\mathrm{SIN}(z) + b\cdot\mathrm{COS}(z) \rightarrow \sqrt{(a^2 + b^2)}\cdot\mathrm{SIN}\left[\frac{\pi\cdot\mathrm{SIGN}(b)}{2} - \mathrm{ATAN}\left[\frac{a}{b}\right] + z\right]$$

Select the Auto direction to apply only those trigonometric transformations that clearly yield a cancellation. Since it usually gives the most compact and attractive form, the initial direction is Auto.

Routine use of either the Expand or Collect direction may produce huge results and take a long time to simplify. However, you may find examples where one direction enables a cancellation that does not occur in the Auto direction.

Exercise: Expand various trigonometric expressions involving angle sums and multiple angles. Start with the expression SIN($2x + 3y$). Then collect the expanded results to see if the original compact expression can be restored.

Use the Toward field of the **Manage Trigonometry** command to control the transformation of powers of sines into cosines and vice versa. When a new direction is selected, the command generates an expression of the form

Trigpower := *direction*

where *direction* is Auto, Sines, or Cosines. The trigonometry power transform direction can also be changed by entering on the author line an expression of the above form using upper and lower case exactly as shown.

Select toward Sines to transform powers of cosines *into* sines using the transformation

$$\mathrm{COS}(z)^2 \rightarrow 1 - \mathrm{SIN}(z)^2$$

Select toward Cosines to transform powers of sines *into* cosines using the transformation

$$\mathrm{SIN}(z)^2 \rightarrow 1 - \mathrm{COS}(z)^2$$

Select Auto to only apply either of the above transformations if it clearly yields a cancellation. Since it usually gives the most compact and attractive form, the initial mode is Auto.

Routine use of either toward Sines or Cosines may produce huge results and take a long time to simplify. However, you may find examples where one or the other enables a cancellation that does not occur in Auto mode.

Use the Angle field of the **Manage Trigonometry** command to specify the unit of angular measurement (degree or radian) used by the trigonometric functions and returned by the inverse trigonometric functions (see Section 6.4). The angular unit setting is saved in ***DERIVE*** initialization files (see Section 2.14). The initial angular unit is radian.

When a new unit of angular measurement is selected, the **Manage Trigonometry** command generates an expression of the form

Angle := *unit*

where *unit* is Degree or Radian. The angular unit can also be changed by entering on the author line an expression of the above form using upper and lower case exactly as shown.

For example, in radian mode SIN(π/4) simplifies to $\sqrt{2}/2$ and ASIN(1/2) simplifies to $\pi/6$ radians. In degree mode SIN(45) simplifies to $\sqrt{2}/2$ and ASIN(1/2) simplifies to 30 degrees. Note that the current angle mode also affects both 2D and 3D plots (see Chapter 5).

If you want cosecants and secants in simplified results, use the **Manage Substitute** command to replace reciprocals of sines and cosines by cosecants and secants. For example, substituting $3\,y\,\mathrm{CSC}(x)$ for $3\,y/\mathrm{SIN}(x)$ in the expression

$$\frac{3 \cdot y}{\mathrm{SIN}(x)} + 7$$

yields the equivalent expression

$$3 \cdot y \cdot \mathrm{CSC}(x) + 7$$

6.4 Inverse Trigonometric Functions

The inverse trigonometric functions return an angle measured in the angular unit (degree or radian) specified by the Angle field of the **Manage Trigonometry** command (see Section 6.3). The following discussion assumes the angular unit is a radian.

- ATAN(z) is the principal arctangent of z (i.e. the angle whose tangent is z). ATAN(y, x) is the angle of the point (x, y) in the x-y plane measured counterclockwise from the positive x-axis. ATAN(y, x) function simplifies to an equivalent expression using the single-argument ATAN function.

  Note that the order of the arguments to ATAN(y, x) is opposite to the usual coordinate pair order. You can remember this by noting that ATAN(y, x) equals ATAN(y/x) if x is positive. If x and y are real, then

$$-\frac{\pi}{2} \le \text{ATAN}(x) \le \frac{\pi}{2}$$

$$-\pi < \text{ATAN}(y, x) \le \pi$$

- ACOT(z) is the principal arccotangent of z (i.e. the angle whose cotangent is z). ACOT(x, y) is the angle of the point (x, y). ACOT(z) simplifies to

$$\frac{\pi}{2} - \text{ATAN}(z)$$

  and ACOT(x, y) simplifies to ATAN(y, x), which is further transformed as described above. If x and y are real, then

$$0 \le \text{ACOT}(x) \le \pi$$

$$-\pi < \text{ACOT}(x, y) \le \pi$$

- ASIN(z) is the principal arcsine of z. Internally ASIN(z) is replaced by the equivalent form

$$\text{ATAN}\left[\frac{z}{\sqrt{(1 - z^2)}}\right]$$

  If $-1 \le x \le 1$, then

$$-\frac{\pi}{2} \le \text{ASIN}(x) \le \frac{\pi}{2}$$

- ACOS(z) is the principal arccosine of z. ACOS(z) simplifies to

$$\frac{\pi}{2} - \texttt{ASIN(z)}$$

which is further transformed as described above. If $-1 \leq x \leq 1$, then

$$0 \leq \texttt{ACOS(x)} \leq \pi$$

- ASEC(z) is the arcsecant of z. ASEC(z) simplifies to ACOS(1/z), which is further transformed as described above.

- ACSC(z) is the arccosecant of z. ACSC(z) is replaced by ASIN(1/z), which is further transformed as described above.

Exercise: Plot the six inverse trigonometric functions and note the domain for which they are real-valued and their corresponding range.

6.5 Hyperbolic Functions

The hyperbolic functions simplify to equivalent expressions involving exponentials.

- SINH(z) is the hyperbolic sine of z. SINH(z) simplifies to

$$\frac{\hat{e}^{z}}{2} - \frac{\hat{e}^{-z}}{2}$$

- COSH(z) is the hyperbolic cosine of z. COSH(z) simplifies to

$$\frac{\hat{e}^{z}}{2} + \frac{\hat{e}^{-z}}{2}$$

- TANH(z) is the hyperbolic tangent of z. TANH(z) simplifies to

$$\frac{\hat{e}^{2\cdot z} - 1}{\hat{e}^{2\cdot z} + 1}$$

- COTH(z) is the hyperbolic cotangent of z. COTH(z) simplifies to

$$\frac{\hat{e}^{2\cdot z} + 1}{\hat{e}^{2\cdot z} - 1}$$

- SECH(z) is the hyperbolic secant of z. SECH(z) simplifies to

$$\frac{2\cdot\hat{e}^{z}}{\hat{e}^{2\cdot z} + 1}$$

- CSCH(z) is the hyperbolic cosecant of z. CSCH(z) simplifies to

$$\frac{2\cdot\hat{e}^{z}}{\hat{e}^{2\cdot z} - 1}$$

6.6 Inverse Hyperbolic Functions

The inverse hyperbolic functions simplify to equivalent expressions involving logarithms.

- ASINH(z) is the inverse hyperbolic sine of z. ASINH(z) simplifies to

$$\mathrm{LN}(\sqrt{(z^2 + 1)} + z)$$

- ACOSH(z) is the inverse hyperbolic cosine of z. ACOSH(z) simplifies to

$$2 \cdot \mathrm{LN}(\sqrt{(z + 1)} + \sqrt{(z - 1)}) - \mathrm{LN}(2)$$

- ATANH(z) is the inverse hyperbolic tangent of z. ATANH(z) simplifies to

$$\frac{\mathrm{LN}(z + 1)}{2} - \frac{\mathrm{LN}(1 - z)}{2}$$

- ACOTH(z) is the inverse hyperbolic cotangent of z. ACOTH(z) simplifies to

$$\frac{\mathrm{LN}\left[\frac{z + 1}{z - 1}\right]}{2}$$

- ASECH(z) is the inverse hyperbolic secant of z. ASECH(z) simplifies to

$$2 \cdot \mathrm{LN}\left[\sqrt{\left[\frac{z + 1}{z}\right]} + \sqrt{\left[\frac{1 - z}{z}\right]}\right] - \mathrm{LN}(2)$$

- ACSCH(z) is the inverse hyperbolic cosecant of z. ACSCH(z) simplifies to

$$\mathrm{LN}\left[\frac{z \cdot \sqrt{\left[\frac{z^2 + 1}{z^2}\right]} + 1}{z}\right]$$

Exercise: Plot the six inverse hyperbolic functions and note the domain for which they are real-valued and their corresponding range.

6.7 Piecewise Continuous Functions

This section describes the functions used for defining functions in a piecewise manner and how these functions handle real arguments. The next section describes how they handle complex arguments. Note that the rules used to simplify expressions involving these functions take into account the user-declared domain of variables (see Section 4.10).

- ABS(x) simplifies to the absolute value of x. An absolute value can be entered and is displayed using vertical bars to delimit the argument. Hence if x is 0 or a positive number, $|x|$ simplifies to x. If x is a negative number, $|x|$ simplifies to $-x$.

- SIGN(x) simplifies to the sign of x. Hence if x is a positive number, SIGN(x) simplifies to 1. If x is a negative number, SIGN(x) simplifies to -1. Since the sign of 0 is indeterminate, SIGN(0) simplifies to ± 1.

The functions MIN and MAX can take any number of arguments, a vector, or a matrix. If given a vector (see Chapter 8), they are applied to the elements of the vector. If given a matrix, they are applied to the rows of the matrix, and the results are returned as a vector.

- MAX(x_1, x_2, ..., x_n) simplifies to the maximum value of its arguments. MAX(x, y) simplifies to the equivalent expression

$$\frac{|x - y|}{2} + \frac{x + y}{2}$$

- MIN(x_1, x_2, ..., x_n) simplifies to the minimum value of its arguments. MIN(x, y) simplifies to the equivalent expression

$$\frac{x + y}{2} - \frac{|x - y|}{2}$$

- STEP(x) is the Heaviside unit step function that simplifies to 1 if $x > 0$ and to 0 if $x < 0$. STEP(x) simplifies to the equivalent expression

$$\frac{\mathrm{SIGN}(x)}{2} + \frac{1}{2}$$

- CHI(a, x, b) is the piecewise indicator function. If $a < x < b$, CHI(a, x, b) simplifies to 1. If $x < a < b$ or $a < b < x$, it simplifies to 0. If $a > b$, it simplifies to –CHI(b, x, a). a defaults to 0 and b defaults to 1. If the relation between a, b, and x cannot be determined, CHI(a, x, b) simplifies to the equivalent expression

$$\frac{\text{SIGN}(x - a)}{2} - \frac{\text{SIGN}(x - b)}{2}$$

- FLOOR(m, n) simplifies to the greatest integer less than or equal to m/n. Since n defaults to 1, FLOOR(m) simplifies to the integer-part of m. For example, FLOOR(5.73) simplifies to 5.

- MOD(m, n) simplifies to m modulo n (i.e. the nonnegative remainder of m/n). MOD(m, 0) simplifies to m. Since n defaults to 1, MOD(m) simplifies to the fractional-part of m. If m or n are not numbers, MOD(m, n) simplifies to the equivalent expression

$$m - n \cdot \text{FLOOR}(m, n)$$

- MODS(m, n) simplifies to the symmetric m modulo n which will be in the half-open interval [$-n/2$, $n/2$). MODS(m, 0) simplifies to m. n defaults to 1. If m or n are not numbers, MODS(m, n) simplifies to the equivalent expression

$$m - n \cdot \text{FLOOR}(m + n/2, n)$$

  Note that if m, n, and p are integers with $n>0$ and $p>0$, MOD(m^p,n) and MODS(m^p,n) are computed efficiently using the power mod algorithm, rather than by computing m^p and then computing the result modulo n.

6.8 Complex Variable Functions

The rules used to simplify expressions involving the complex variable functions take into account the current branch selection (see Section 4.5) and the user-declared domain of variables (see Section 4.10).

- #i is the imaginary unit, $\sqrt{(-1)}$. On PC compatible computers #i displays as $\hat{\imath}$ and can be entered by pressing **Alt-I**. You are free to use the unembellished variable name i to represent electrical current, interest, etc.

- unit_circle represents some arbitrary point on the unit circle in the complex plane. For instance, 1, −1, $\hat{\imath}$, and $-\hat{\imath}$ are points on the unit circle. unit_circle may arise in limit problems or when solving equations. For example, if z is declared complex (see Section 4.10), solving the equation $|z| = 2$ for z gives $z = 2 \cdot$unit_circle.

- ABS(z) simplifies to the absolute value (also called magnitude or modulus) of z. The absolute value of z is the distance between z and the origin of the complex plane. An absolute value can be entered and is displayed using vertical bars to delimit the argument. Hence if x and y are real, $|x + \hat{\imath}y|$ simplifies to

$$\sqrt{(x^2 + y^2)}$$

- SIGN(z) simplifies to the point on the unit circle in the complex plane that has the same phase angle as z. For example, SIGN(3+4$\hat{\imath}$) simplifies to 3/5+4/5$\hat{\imath}$.

- RE(z) simplifies to the real part of z. Hence if x and y are real, RE($x + \hat{\imath}y$) simplifies to x.

- IM(z) simplifies to the imaginary part of z. Hence if x and y are real, IM($x + \hat{\imath}y$) simplifies to y.

- CONJ(z) simplifies to the complex conjugate of z. Hence if x and y are real, CONJ($x + \hat{\imath}y$) simplifies to $x - \hat{\imath}y$.

- PHASE(z) simplifies to the principal phase angle of z, measured counterclockwise from the positive x-axis. The angle is measured in the angular unit (degree or radian) specified by the Angle field of the **Manage Trigonometry** command (see Section 6.3). Thus if the angular unit is a radian, it is the case that

$$-\pi < \text{PHASE}(z) \leq \pi$$

  for any z in the complex plane.

6.9 Probability Functions

The functions in this section are useful for solving probability problems.

- euler_gamma is **Euler's constant** (0.577215...). In exact mode euler_gamma remains symbolic; otherwise it simplifies to an approximate value accurate to the current precision.

- $z!$ is the factorial of z. If n is a positive integer, $n!$ is the product of the first n integers. For example, 5! simplifies to 120. The factorial operator (!) is also defined for real and complex arguments. For example, (3/2)! simplifies to

$$\frac{3 \cdot \sqrt{\pi}}{4}$$

Exercise: Use the **Factor** command to factor $n!$ for n equal 50, 60, 70, and 80. Make a conjecture about the exponents of the factors and test it out on some additional integers.

- GAMMA(z) is the gamma function of z. GAMMA displays as Γ and can be entered by pressing **Alt-G** on PC compatible computers. GAMMA(z) simplifies to $(z - 1)!$. The incomplete gamma function and its relatives are defined in PROBABIL.MTH (see Section 9.12).

- PERM(m, n) is the number of permutations or arrangements of m things taken n at a time. PERM(z, w) simplifies to

$$\frac{z!}{(z - w)!}$$

- COMB(m, n) is the number of combinations or groupings of m things taken n at a time. Note that COMB(m, n) is also the binomial coefficient

$$\binom{m}{n}$$

COMB(z, w) simplifies to

$$\frac{z!}{w! \cdot (z - w)!}$$

Exercise: Build a table of the first ten rows of Pascal's triangle of binomial coefficients.

> **Hint**: Use COMB(m, n) in conjunction with two VECTOR commands to independently vary m and n from 0 to 9 (see Section 8.2).

- RANDOM(n) invokes a pseudo-random number generator. If n is an integer, then

  if $n>1$, RANDOM(n) returns a random *integer* in the interval [0, n). For example, each time RANDOM(6) is simplified, it returns with equal probability one of the 6 integers from the set {0, 1, 2, 3, 4, 5}.

  if $n=1$, RANDOM(n) returns a random *number* in the interval [0, 1).

  if $n<0$, RANDOM(n) initializes the random number seed to $-n$.

  if $n=0$, RANDOM(n) initializes the random number seed based on the time in centiseconds since ***DERIVE*** was started and returns that time.

The sum of the throw of a pair of dice is an integer between 2 and 12 inclusive. The probabilities of obtaining each of these integers are not equal. For example, the probability of obtaining 2 is 1/36, whereas the probability of obtaining 7 is 1/6.

Exercise: Assign the variable dice a value such that each time dice is simplified it returns an integer between 2 and 12 inclusive with the same probabilities as for dice throwing.

> **Hint**: The throw of a single die would be defined by the assignment
>
> ```
> die := RANDOM(6) + 1
> ```

Then "throw" the variable dice a 100 times using the command

```
VECTOR (dice, n, 1, 100)
```

and see if 2 and 7 occur with roughly the correct probabilities.

6.10 Statistical Functions

The functions in this section are useful for solving statistical problems. The first four can take any number of arguments, a vector, or a matrix. If given a vector (see Chapter 8), the function is applied to the elements of the vector. If given a matrix, the function is applied to the rows of the matrix, and the results are returned as a vector.

- AVERAGE(z_1, z_2, ..., z_n) is the arithmetic mean or average of z_1, z_2, ..., z_n. AVERAGE(z_1, z_2, ..., z_n) simplifies to

$$\frac{z_1 + z_2 + \ldots + z_n}{n}$$

  For example, both the expressions AVERAGE(2, 3, 4) and AVERAGE([2, 3, 4]) simplify to 3.

- RMS(z_1, z_2, ..., z_n) is the root mean square of z_1, z_2, ..., z_n. RMS(z_1, z_2, ..., z_n) simplifies to

$$\sqrt{\frac{z_1^2 + z_2^2 + \ldots + z_n^2}{n}}$$

  For example,

$$\text{RMS}\begin{bmatrix} 3 & 4 & 5 \\ 5 & 12 & 13 \end{bmatrix}$$

  simplifies to

$$\left[\frac{5 \cdot \sqrt{6}}{3}, \frac{13 \cdot \sqrt{6}}{3}\right]$$

- VAR(z_1, z_2, ..., z_n) is the unbiased sample variance of z_1, z_2, ..., z_n. VAR(z_1, z_2, ..., z_n) simplifies to

$$\frac{(z_1 - a)^2 + (z_2 - a)^2 + \ldots + (z_n - a)^2}{n - 1}$$

  where a is the average of z_1 through z_n.

- STDEV(z_1, z_2, ..., z_n) is the sample standard deviation of z_1, z_2, ..., z_n. STDEV(z_1, z_2, ..., z_n) simplifies to

$$\sqrt{\text{VAR}}(z_1, z_2, \ldots, z_n)$$

which is further transformed as described for the function VAR.

- FIT(v, A) returns the least squares fit of a parameterized expression in label vector v to the set of points in the data matrix A. A **least squares fit** minimizes the sum of the squares of the discrepancies at the points. Chapter 8 describes vectors and matrices.

The elements of the label vector are the data variables followed by the parameterized expression. The expression should depend on the data variables *and* one or more parametric variables. The dependence on the parametric variables should be linear (i.e. if p is a parametric variable, the expression must be of the form $r \cdot p + s$, where r and s are expressions independent of p). The expression's dependence on the data variables need not be linear.

The elements of each row of the data matrix are the numeric values of the data variables and the corresponding numeric value of the expression given in the label vector. The data values on each row must correspond one-to-one with the elements of the label vector.

When the number of data matrix rows *equals* the number of parametric variables, FIT returns an expression that exactly fits the data, to within roundoff error. For example,

$$\text{FIT}\left([x,\ a \cdot x^2 + b \cdot x + c],\ \begin{bmatrix} -1.5 & 0 \\ 0.5 & -2 \\ 1.5 & -1.5 \end{bmatrix}\right)$$

simplifies to the parabola

$$\frac{x^2}{2} - \frac{x}{2} - \frac{15}{8}$$

When the number of data matrix rows *exceeds* the number of parametric variables, FIT returns an expression that is a least squares fit to the data. For example,

$$\text{FIT}\left([x,\ y,\ a \cdot x + b \cdot y + c],\ \begin{bmatrix} 2.75 & -2.3 & 2.4 \\ -3.5 & 4.5 & 4.2 \\ 5 & 3.5 & 5.8 \\ -4 & -5 & 1.3 \end{bmatrix}\right)$$

approximates to the plane

```
0.153644·x + 0.357749·y + 3.35279
```

Note that the dependence on the data variables need not be linear. For example,

```
FIT [[t, q·ATAN(t) + r·SIN(t)], [-4  -1  ]]
    [                           [-1  -1.9]]
    [                           [ 2   2.25]]
```

approximates to

```
1.29723·ATAN(t) + 0.961378·SIN(t)
```

Exercise: Highlight and plot this approximate result. Then highlight and plot the original data matrix. Note how closely the approximate result fits the data points.

Even though the dependence on the parametric variables in a parameterized expression is nonlinear, it may be possible to transform the expression so the dependence is linear.

For example, to fit $a \cdot \hat{e}^{b \cdot x}$ to a set of data points, first transform it to $\ln(a)+b \cdot x$ by taking its natural logarithm. Then if $\ln(a)$ is replaced by a' in the transformed expression, FIT can be used to find the a' and b that give the best fit of the transformed expression to the natural logarithm of the original data points. Finally, approximate $\hat{e}^{a'}$ to find a.

6.11 Error and Zeta Functions

The error and Riemann zeta functions frequently occur in probability and statistics problems.

- ERF(z) is the integral of the Gaussian distribution from 0 to z. It is defined as

$$\frac{2}{\sqrt{\pi}} \cdot \int_0^z \hat{e}^{-t^2} \, dt$$

- ERF(z, w) is the generalized error function. ERF(z, w) is defined as the integral of the Gaussian distribution from z to w. ERF(z, w) simplifies to

```
ERF(w) - ERF(z)
```

- ERFC(z) is the complementary error function. ERFC(z) simplifies to

```
1 - ERF(z)
```

- NORMAL(z, m, s) is the normal distribution function of z with mean m and standard deviation s. NORMAL(z, m, s) simplifies to

$$\frac{\text{ERF}\left[\frac{\sqrt{2} \cdot z}{2 \cdot s} - \frac{\sqrt{2} \cdot m}{2 \cdot s}\right] + 1}{2}$$

  with m defaulting to 0 and s defaulting to 1. This makes NORMAL(z) equivalent to the cumulative distribution function

$$\Phi(z) = \frac{1}{\sqrt{(2 \cdot \pi)}} \cdot \int_{-\infty}^z \hat{e}^{-t^2/2} \, dt$$

- ZETA(s) is the Riemann zeta function of s. ZETA(s) simplifies to an exact closed form value if one is known. For example, ZETA(2) simplifes to $\pi^2/6$. ZETA(s) approximates to a numerical value if s is a real or complex number. Functions for approximating several generalized zeta functions are defined in the utility file ZETA.MTH (see Section 9.18).

  Note that ***DERIVE*** simplifes the infinite sum

$$\sum_{k=1}^{\infty} k^{-s}$$

  to ZETA(s), if s is a number greater than 1 (see Section 7.5).

6.12 Financial Functions

An **annuity** is a loan, mortgage, or similar contract to make fixed payments of money at regular intervals over a certain period of time in exchange for a sum of money. Interest is paid on the money at a fixed rate and is compounded each interval.

DERIVE's financial functions are based on the equation

$$v\cdot(1+i)^n + p\cdot(1+i\cdot t)\frac{(1+i)^n - 1}{i} + f = 0$$

that applies to an annuity with a present value v in which a fixed payment p is paid for n regular periods at a periodic interest rate of i with a future value of f. Positive numbers represent money you receive; negative numbers represent money you pay out.

The financial functions have an optional fifth argument t, the time of payment. Make this argument 0 if the payments are made at the *end* of each period, 1 if the payments are made at the *beginning* of each period, or a proportionate fraction between 0 and 1 if the payments are made *within* each period. The default value of t is 0, indicating that payments are made at the *end* of each period.

Because of catastrophic cancellation in the above formula, the default precision of 6 significant digits is insufficient for many financial calculations, even if less accuracy than that is needed in the result. Thus it is advisable to use 10 or more significant digits of precision when using these functions (see Section 3.8).

- PVAL(i, n, p, f, t) is the present value of the contract. The future value argument f defaults to 0.

- FVAL(i, n, p, v, t) is the future value of the contract. The present value argument v defaults to 0.

- PMT(i, n, v, f, t) is the periodic payment of the contract. The present value argument v and the future value argument f default to 0.

- NPER(i, p, v, f, t) is the number of payment periods. The present value argument v and the future value argument f default to 0.

- RATE(n, p, v, f, t) is the periodic interest rate. Note that RATE returns the interest rate per period, not necessarily the *annual* interest rate. The present value argument v and the future value argument f default to 0.

For example, to compute the monthly payment on a 4 year car loan of $15,000 at a 10% annual interest rate, approximate

```
PMT(10%/12, 4·12, 15000)
```

using 10 significant digits of precision. This gives a monthly payment of −380.4387515. The amount is negative because it represents an outflow of money.

As another example, to compute the balance of a savings account in 5 years after making an initial deposit of $1,000, making monthly payments of $100 at the *beginning* of the month, and earning 5.5% annual interest compounded monthly, approximate

```
FVAL(5.5%/12, 5·12, −100, −1000, 1)
```

using 10 significant digits of precision. This gives a future balance of 8235.356459. The monthly payment and the initial deposit are negative because they represent an outflow of money.

Exercise: Approximate

```
RATE(36, −300, 9000)
```

to determine the effective *monthly* rate of interest on a 36 month $9000 loan for which payments of $300 are made at the *end* of the month. Compare this with the rate if payments are made at the *beginning* of the month.

6.13 Number Theory Functions

The pre-defined functions in this section are useful for investigations in number theory. Additional functions for this purpose are defined in the utility file NUMBER.MTH (see Section 9.20).

The functions GCD and LCM can take any number of arguments, a vector, or a matrix. If given a vector (see Chapter 8), they are applied to the elements of the vector. If given a matrix, they are applied to the rows of the matrix, and the results are returned as a vector.

- GCD(m_1, m_2, ..., m_n) simplifies to the greatest common divisor of the numbers m_1, m_2, ..., m_n. For example, GCD(6, 15) simplifies to 3.

- LCM(m_1, m_2, ..., m_n) simplifies to the least common multiple of the numbers m_1, m_2, ..., m_n. For example, LCM([6, 15, 14]) simplifies to 210. If m or n are not numbers, LCM(m, n) simplifies to the equivalent expression

$$\frac{|m \cdot n|}{\text{GCD}(n,\ m)}$$

- NUMBER(m) returns true if its argument simplifies to a number (either integer or rational); otherwise it returns false. NUMBER is a predicate useful in the conditional clauses of IF and SELECT constructs (see Section 10.3). For example,

```
NUMBER(SQRT(m^2 + n^2))
```

  returns true if the sum of the squares of m and n is a perfect square.

- PRIME(m) returns true or false based on whether or not m is a prime number. PRIME is a predicate useful in the conditional clauses of IF and SELECT constructs (see Section 10.3). It uses a probabilistic Monte Carlo primality test. If n is a positive integer, PRIME(m,n) performs the primality test n times before before m is assumed prime. The default value of n is 6.

- NEXT_PRIME(m) is the next prime number larger than the number m. For example, NEXT_PRIME(1000) simplifies to 1009. It uses a probabilistic Monte Carlo primality test. If n is a positive integer, NEXT_PRIME(m,n) performs the primality test n times before a number is assumed prime. The default value of n is 6.

6.14 Expression Decomposition Functions

The pre-defined functions in this section are useful for decomposing expressions into their component parts. Some of the functions require that their arguments be polynomials or numbers.

- NUMERATOR(z) returns the numerator of z. z can be a rational number or a symbolic expression. For example,

$$\text{NUMERATOR}\left[\frac{3\cdot x^2 - 5\cdot x + 7}{x - 4}\right]$$

simplifies to

$$3\cdot x^2 - 5\cdot x + 7$$

- DENOMINATOR(z) returns the denominator of z. z can be a rational number or a symbolic expression.

Before calling NUMERATOR or DENOMINATOR you can use FACTOR and specify a Trivial amount of factoring to put an expression over a common denominator (see Section 4.6).

- QUOTIENT(u,v) returns the quotient of u divided by v. u and v should be rational numbers or polynomials. For example,

$$\text{QUOTIENT}(x^4 + 3\cdot x^3 + 5\cdot x + 6,\ x^2 - 5)$$

simplfies to

$$x^2 + 3\cdot x + 5$$

- REMAINDER(u,v) returns the remainder of u divided by v. u and v should be rational numbers or polynomials.

$$\text{REMAINDER}(x^4 + 3\cdot x^3 + 5\cdot x + 6,\ x^2 - 5)$$

simplfies to

$$20\cdot x + 31$$

- POLY_GCD(u,v) returns the gcd (greatest common divisor) of u and v. u and v should be polynomials. For example,

$$\text{POLY\_GCD}(x^3 + 3\cdot x^2 + 5\cdot x + 6,\ x^3 + 2\cdot x - 3)$$

  simplifies to

$$x^2 + x + 3$$

- TERMS(u) returns a vector of the syntactic terms of u. Before calling TERMS you can use EXPAND to expand products and powers to sums (see Section 4.4). TERMS distributes over the elements of a vector.

- FACTORS(u) returns a vector of the syntactic factors of u. Before calling FACTORS you can use FACTOR to factor a sum (see Section 4.6). FACTORS distributes over the elements of a vector.

- VARIABLES(u) returns a vector of the free variables of u ordered from most to least main (see Section 4.3).

- LHS(u) returns the left hand side of an equation or relation. For example, LHS($2x+3=5$) simplifies to $2x+3$. If u is a vector, LHS returns a vector consisting of the left hand sides of u's elements.

- RHS(u) returns the right hand side of an equation or relation. For example, RHS($2x+3=5$) simplifies to 5. If u is a vector, RHS returns a vector consisting of the right hand sides of u's elements. For example,

$$\text{RHS}(\text{SOLVE}(x\hat{}2 - 5\cdot x + 6 = 0,\ x))$$

  simplifies to the vector [2, 3] (i.e. the roots of the equation).

Chapter 7. Calculus

DERIVE provides commands and functions for doing symbolic limits, differentiation, Taylor polynomial approximations, integration, series summation, and extended products. Skip this chapter if you do not plan to use these commands and functions. To see a demonstration of the built-in calculus functions, load the file CALCULUS.DMO using a **Transfer Demo** command (see Section 2.9).

Included with ***DERIVE*** are several utility MTH files that define additional calculus functions in terms of the built-in functions (see Chapter 9).

The **Calculus** commands do *not* automatically simplify expressions unless you press **Ctrl-Enter** or **Ctrl-J** instead of **Enter** when issuing these commands.

7.1 Limits

Use the **Calculus Limit** command to find the limit of an expression as one of its variables approaches a point. After you select an expression, this command prompts you to enter the limit variable. Then it prompts you to enter the limit point and the direction from which the point is approached on the real line.

Alternatively, the limit of the expression u as the variable x approaches the point a can be entered by authoring the form LIM(u, x, a). a defaults to 0. For example,

```
LIM(((x + h)^2 - x^2)/((x + h) - x), h, 0)
```

displays as

$$\lim_{h \to 0} \frac{(x + h)^2 - x^2}{x + h - x}$$

and simplifies to $2x$. Note that this result is the derivative of x^2.

If there is a discontinuity in u at $x=a$, the limit as x approaches a from the right may differ from the limit as x approaches a from the left. The right-hand limit can be entered by calling LIM with a positive number, such as 1, as its fourth argument. For example,

```
LIM(SIGN(x), x, 0, 1)
```

displays as

$$\lim_{x \to 0+} \text{SIGN}(x)$$

and simplifies to 1.

The left-hand limit can be entered by calling LIM with a negative number, such as −1, as its fourth argument. For example,

```
LIM(SIGN(x), x, 0, -1)
```

displays as

$$\lim_{x \to 0-} \text{SIGN}(x)$$

and simplifies to −1. Note that the two-sided limit

$$\lim_{x \to 0} \text{SIGN}(x)$$

simplifes to ± 1.

As another example, the right-hand limit

$$\lim_{x \to 0+} \frac{1}{x}$$

simplifies to ∞ (plus infinity), the left-hand limit

$$\lim_{x \to 0-} \frac{1}{x}$$

simplifies to $-\infty$ (minus infinity), and the two-sided limit

$$\lim_{x \to 0} \frac{1}{x}$$

simplifies to $\pm\infty$.

Infinity can be entered as a limit point. Use *inf* for plus infinity and *−inf* for minus infinity (see Section 4.9). For example,

```
LIM(a·x/(x + 1), x, inf)
```

displays as

$$\lim_{x \to \infty} \frac{a \cdot x}{x + 1}$$

and simplifies to a.

Exercise: Find the limit of

$$\frac{1}{x \cdot (\hat{e}^{1/x} + 1)}$$

as x approaches 0 from the left and from the right, and as x approaches plus and minus infinity. Confirm your results by plotting the expression in a 2D-plot window (see Section 5.1).

The **Calculus Limit** command or LIM function can be regarded as a more powerful version of the **Manage Substitute** command: The limit of an expression may simplify to a unique value, whereas substitution in the same expression may simplify to an ambiguous answer such as $\pm\infty$ (plus-or-minus infinity), ± 1 (plus-or-minus one), or ? (unknown). For example,

$$\lim_{x \to 0} \frac{x}{\mathrm{SIN}(x)}$$

simplifies to 1, whereas substituting 0 for x yields

$$\frac{0}{\mathrm{SIN}(0)}$$

which simplifies to ? .

To facilitate using LIM for substitution, its second argument can be a vector of variables, instead of just a variable. Then its third argument should be a vector of values for those variables. For example,

```
LIM(x^2 + y^2, [x, y], [2, 3])
```

simplifies to $13 = 2^2+3^2$.

You can use LIM to substitute values for variables in function definitions (see Section 4.12). If u is an expression involving x,

$$\frac{u(b) - u(a)}{b - a}$$

is the slope of the secant line through $u(a)$ and $u(b)$. Thus, the following is a definition useful for finding secant lines:

$$\mathrm{SLOPE}(u, x, a, b) := \frac{\lim_{x \to b} u - \lim_{x \to a} u}{b - a}$$

Exercise: Define SLOPE as shown and find the slope of the secant line of EXP(x) through the points $x=1$ and $x=2$.

7.2 Differentiation

Use the **Calculus Differentiate** command to find the partial derivative of an expression. After you select an expression, this command prompts you to enter the name of the differentiation variable and the order of the derivative.

Alternatively, the partial derivative of the expression u with respect to the variable x can be entered by authoring the form DIF(u, x). For example,

```
DIF(SIN(a·x^2), x)
```

displays as

$$\frac{d}{dx}\,\mathrm{SIN}(a\cdot x^2)$$

and simplifies to

$$2\cdot a\cdot x\cdot \mathrm{COS}(a\cdot x^2)$$

The nth-order partial derivative of the expression u with respect to the variable x can be entered by authoring the form DIF(u, x, n). For example, the second-order partial derivative

```
DIF(LN(COS(x)), x, 2)
```

displays as

$$\left[\frac{d}{dx}\right]^2 \mathrm{LN}(\mathrm{COS}(x))$$

and simplifies to

$$-\mathrm{TAN}(x)^2 - 1$$

You can issue successive **Calculus Differentiate** commands or enter nested calls on DIF to obtain mixed partial derivatives. For example, the mixed second-order partial derivative

```
DIF(DIF((a·x + b·y)^3, x), y)
```

displays as

$$\frac{d}{dy}\,\frac{d}{dx}\,(a\cdot x + b\cdot y)^3$$

and simplifies to

$$6\cdot a\cdot b\cdot (a\cdot x + b\cdot y)$$

The derivative of an expression simplifies to 0 if the expression does not explicitly contain the differentiation variable. To take the derivative of a general function, first use the **Declare Function** command to declare an arbitrary function (see Section 4.12). For example if F has been declared an arbitrary function,

$$\frac{d}{dx}\ F(x)^3$$

simplifies to

$$3 \cdot F(x)^2\ \frac{d}{dx}\ F(x)$$

DERIVE uses the chain rule to differentiate functions of functions. Differentiation in ***DERIVE*** must be with respect to a variable and *not* a function. Thus a new variable named *@n* is introduced when differentiating an arbitrary function of an expression or function. For example if F has been declared an arbitrary function

$$\frac{d}{dx}\ F(x^2)$$

simplifies to

$$2 \cdot x \cdot \lim_{@1 \to x^2} \frac{d}{d\ @1}\ F(@1)$$

DERIVE can compute the derivative of all its pre-defined mathematical functions and operators with the exception of the factorial operator ! (see Section 6.9). The derivative of ! can be defined in terms of the psi and polygamma functions defined in the utility file PROBABIL.MTH (see Section 9.12).

Functions for numerical differentiation are defined in NUMERIC.MTH (see Section 9.3). Many applications of differentiation, including functions for implicit differentiation, curvature, tangents, and osculating circles, are defined in DIF_APPS.MTH (see Section 9.4).

7.3 Taylor Polynomial Approximations

Use the **Calculus Taylor** command to find a Taylor polynomial approximation to an expression. After you select an expression, this command prompts you to enter the name of the expansion variable. Then it prompts you to enter the order (maximum degree) of the approximation and the expansion point.

Alternatively, the Taylor polynomial approximation of the expression u with respect to the expansion variable x about the expansion point a of order n can be entered by authoring the form TAYLOR(u, x, a, n). For example,

```
TAYLOR(ê^x, x, 0, 5)
```

simplifies to

$$\frac{x^5}{120} + \frac{x^4}{24} + \frac{x^3}{6} + \frac{x^2}{2} + x + 1$$

The **order** of the approximation is the number of partial derivatives used in the truncated Taylor series approximating the expression. Some of these derivatives may simplify to 0 at the expansion point, so the number of terms and the degree of the resulting polynomial may be less than the specified order. For example,

```
TAYLOR(c·SIN(x), x, 0, 6)
```

simplifies to

$$\frac{c \cdot x^5}{120} - \frac{c \cdot x^3}{6} + c \cdot x$$

For some expressions, successive partial derivatives grow rapidly in size and hence computing time. Thus when computing Taylor polynomial approximations, it is wise to start with a small value for the order of approximation (such as 2 or 3).

Exercise: Compute the first, second, and third derivatives of

$$\frac{x + 1}{EXP(x) + 1}$$

and note the strongly increasing complexity and computing time for successive derivatives.

It is possible for all the derivatives of an expression to evaluate to 0 at a point, even though the expression is *not* a constant. The Taylor polynomial approximation of such an expression expanded about this point is 0, no matter how large the order of approximation.

Exercise: Verify that the 4th, 5th, and 6th order Taylor polynomial approximations of

$$\text{EXP}\left[-\frac{1}{x^2}\right]$$

about the point $x=0$ are all 0. Then plot the above expression to see that 0 is indeed a good approximation of the expression near $x=0$.

Some expressions cannot be approximated by a Taylor polynomial beyond a certain order because the derivatives are not finite at the expansion point beyond that order. In such cases, TAYLOR simplifies to $\pm\infty$, complex infinity, or ? (unknown). For example, the 0th order Taylor polynomial approximation of $\sqrt{x}$ about the point $x=0$ is 0, but the first order approximation is complex ∞ and the second order approximation is ? .

Functions that use Taylor series approximations to solve equations, invert functions, and solve first and second order differential equations are defined in various utility files (see Chapter 9).

7.4 Integration

Use the **Calculus Integrate** command to find the **antiderivative** or a **definite integral** of an expression. After you select an expression, this command prompts you to enter the name of the integration variable, the lower limit, and the upper limit. Note that the *#n* notation (see Section 3.3) can be used to refer to previous expressions when entering limits. If the limit fields are left blank, the antiderivative is returned; otherwise the definite integral is returned.

Alternatively, the antiderivative of the expression u with respect to the variable x can be entered by authoring the form INT(u, x). The definite integral of the expression u with respect to the variable x from the lower limit a to the upper limit b can be entered by authoring the form INT(u, x, a, b).

Antiderivatives and definite integrals are displayed using an integral sign. Upon simplification, *DERIVE* attempts to determine a closed-form expression whose partial derivative is the integrand. A **closed-form expression** does not contain infinite series, infinite products, integrals, or limits. For example,

```
INT(x/(a^2 + b^2·x^4), x)
```

displays as

$$\int \frac{x}{a^2 + b^2 \cdot x^4}\, dx$$

and simplifies to

$$\frac{\text{ATAN}\left[\frac{b \cdot x^2}{a}\right]}{2 \cdot a \cdot b}$$

Use the **Calculus Differentiate** command to find higher-order antiderivatives by entering a *negative* integer for the order of the derivative. Alternatively, the *n*th-order antiderivative of the expression u with respect to the variable x can be entered by authoring the form DIF(u, x, $-n$). For example, the second-order antiderivative

```
DIF(-TAN(x)^2 - 1, x, -2)
```

displays as

$$\left[\frac{d}{dx}\right]^{-2} (-\text{TAN}(x)^2 - 1)$$

and simplifies to

```
LN(COS(x))
```

Issue successive **Calculus Integrate** commands or enter nested calls on INT to obtain iterated definite integrals. This provides a convenient method for computing double integrals and triple integrals. For example, the iterated definite integral

```
INT(INT(x·LN(y), y, x, 2·x), x, 0, 2)
```

displays as

$$\int_0^2 \int_x^{2\cdot x} x\cdot LN(y)\ dy\ dx$$

and simplifies to

$$8\cdot LN(2) - \frac{32}{9}$$

DERIVE uses powerful techniques that enable it to find antiderivatives for a large class of expressions. These expressions include extended polynomials; products of polynomials with sinusoids, logarithms, or exponentials that are linear in their arguments; piecewise continuous functions; and many rational expressions or expressions involving square roots of linear or quadratic expressions.

If a simplified result includes one or more integral signs it is because the antiderivative does not exist *or* ***DERIVE*** is unable to find the antiderivative.

If ***DERIVE*** determines that a closed-form antiderivative does not exist in terms of its pre-defined functions and operators, it displays the message No elementary integral on the message line.

Exercise: Try finding the antiderivative of SIN(x)/x with respect to x.

If ***DERIVE*** is unable to find an antiderivative that does exist, you may be able to help it along by making a change of variable or using integration by parts. Functions to facilitate both these integration techniques are defined in the utility file MISC.MTH (see Section 9.20).

If all else fails, you can approximate the antiderivative of an expression by integrating its Taylor polynomial approximation (see Section 7.3). You may need to use several different expansion points to make the series converge more rapidly over the interval of interest.

Exercise: Approximate the antiderivative of SIN(x)/x by integrating its fourth order Taylor polynomial approximation about the point $x=0$.

As with most integral tables, ***DERIVE*** does *not* add an arbitrary constant to its antiderivatives. To add an arbitrary constant to antiderivatives, use a new constant name each time to prevent unjustified cancellations of the constants when subtracting antiderivatives.

When comparing a ***DERIVE*** produced antiderivative with one in an integral table, it is important to realize that they may appear quite different. They may be different forms of equivalent expressions, or they may differ by a well disguised constant. Note that if such different forms occur, you can use ***DERIVE*** to simplify the difference of the two antiderivatives and see if the result is independent of the integration variable.

Exercise: Declare n positive using the **Declare Variable Domain** command and compute the antiderivative of x^n. Then declare n real and recompute the antiderivative. Why do the antiderivatives differ by the constant $-1/(n+1)$?

> **Hint**: Take the limit of both antiderivatives as n approaches -1 to determine which gives the correct antiderivative of x^n when $n=-1$.

If precision is in approximate mode and the integrand depends only on the integration variable and both limits are numeric, then ***DERIVE*** uses an extrapolated adaptive **Simpson's rule** to numerically approximate definite integrals. This algorithm is also used if precision is in mixed mode and ***DERIVE*** is unable to determine a closed-form antiderivative for the integrand.

The computing time for this **numerical integration** algorithm usually increases sharply with the current precision. For many integrals the resulting accuracy is nearly that of the current precision. However, the accuracy may be significantly worse than the current precision if the integrand or one of its low order derivatives has a discontinuity or a singularity within the interval or at either endpoint. Integrals that have relatively equal areas above and below the axis can also cause poor relative error via catastrophic cancellation. ***DERIVE*** is usually able to recognize that one of these situations has occurred, in which case it displays the warning message

Dubious accuracy

If precision is in exact or mixed mode, ***DERIVE*** simplifies definite integrals by trying to determine a closed-form antiderivative for the integrand. Then it subtracts the limit of the antiderivative as the integration variable approaches the lower limit from the limit as the integration variable approaches the upper limit. Both limits are approached from *within* the interval. Thus, ***DERIVE*** can determine some **improper integrals** having infinite limits or integrand singularities at the limits. For example,

$$\int_1^{\infty} \frac{1}{x^2}\,dx$$

simplifies to 1, and

$$\int_0^1 \frac{1}{\sqrt{x}}\,dx$$

simplifies to 2.

Although this technique is valid for endpoint singularities, it may be invalid if there is a singularity within the integration interval. Instead of the Riemann integral, it may yield the **Cauchy principal value**. For example,

$$\int_{-1}^{2} \frac{1}{x^3}\,dx$$

simplifies to 3/8 which is correct only if the assumption is made that the singularity at 0 is approached at the same rate from both sides, thus exactly canceling the positive and negative infinite areas.

An even more serious problem occurs when a difference of antiderivatives across a singularity gives results that even Cauchy would call wrong. For example,

$$\int_{-1}^{1} \frac{1}{x^2}\,dx$$

simplifies to –2. This is the difference of the antiderivative evaluated at the limit points, but it is obviously not correct since the integrand is positive throughout the interval of integration.

As is the case when using integral tables, it is your responsibility to find singularities that occur within the interval of integration and to split the integrals there. For example,

$$\int_{-1}^{0} \frac{1}{x^2}\,dx + \int_0^1 \frac{1}{x^2}\,dx$$

simplifies to ∞, which is correct.

If the integrand looks as though it may have singularities but their locations are not obvious, try using the **soLve** command to find the zeros of the integrand's common denominator. Plotting the integrand over the integration interval may also help. However, there is no known method that is guaranteed to find all internal singularities.

There may exist a closed-form definite integral of an expression for which there is no closed-form antiderivative. Although ***DERIVE*** can *not* directly integrate such expressions, it is useful for doing the steps required by various advanced techniques required to solve such problems.

Many applications of integration, including functions for Laplace transforms, Fourier series, arc lengths, surface areas, centroids, and moments of inertia, are defined in INT_APPS.MTH (see Section 9.5).

7.5 Sums

The sum of $f(i)$ as i goes from m to n is denoted by

$$\sum_{i=m}^{n} f(i)$$

If n is equal to $m-1$, the above sum is defined to be 0. If n is less than $m-1$, the above sum is defined as

$$\sum_{i=n+1}^{m-1} -f(i)$$

If f and F are functions such that for all n,

$$f(n) = F(n+1) - F(n)$$

then $F(n)$ is called the **antidifference** of $f(n)$. Using telescoping cancellation, it is easy to see that

$$\begin{aligned}\sum_{i=m}^{n} f(i) &= f(m) + f(m+1) + \cdots + f(n-1) + f(n) \\ &= F(n+1) - F(m)\end{aligned}$$

Thus finding the antidifference of an expression enables you to determine closed-form **definite sums** of the expression with variable upper or lower limits. This is analogous to using antiderivatives to compute definite integrals.

Use the **Calculus Sum** command to find the antidifference or the definite sum of an expression. After you select an expression, this command prompts you to enter the name of the summation variable, and then the lower and upper limits. If the limit fields are made blank, the antidifference is returned; otherwise the definite sum is returned.

Alternatively, the antidifference of the expression u with respect to the variable n can be entered by authoring the form SUM(u, n). For example,

```
SUM(1/2^n, n)
```

displays using the usual Σ (Sigma) summation notation as

$$\sum_{n} \frac{1}{2^n}$$

and simplifies to -2^{1-n}.

The definite sum of the expression u with respect to n from k to m can be entered by authoring the form SUM(u, n, k, m). For example,

```
SUM(1/2^n, n, 1, m)
```

displays as

$$\sum_{n=1}^{m} \frac{1}{2^n}$$

and simplifies to $1-2^{-m}$.

To avoid definite sums simplifying to expressions involving the FLOOR function (see Section 6.7), ***DERIVE*** assumes variables occurring in the sums' upper and lower limits are integers. Therefore, substituting noninteger values for these variables may give incorrect results.

As with antiderivatives, closed-form antidifferences may not exist in terms of the operators and functions known to ***DERIVE***. Even when such an antidifference exists, there is no known method that is guaranteed to find it.

DERIVE uses several techniques that enable it to find the antidifference of a large class of expressions. These expressions include extended polynomials, products of polynomials with exponentials that are linear in their arguments, some rational expressions, and telescoping sums.

Of course, if the upper and lower limits are actual numbers, the sum can be computed directly without having to find the antidifference. For example,

```
SUM(1/k^2, k, 1, 5)
```

simplifies to 5269/3600 even though a closed-form antidifference does not exist.

If the third argument of SUM is a vector, the summation variable ranges over the elements of the vector. For example,

```
SUM(k^2, k, [2, 3, 5, 7, 11])
```

simplifies to 208, the sum of the squares of the first five prime numbers.

Finally, if given a single argument which is a vector, SUM returns the sum of the elements. For example,

```
SUM([2, 3, 5, 7, 11])
```

simplifies to 28, the sum of the first five prime numbers.

7.6 Products

The product of $f(i)$ as i goes from m to n is denoted by

$$\prod_{i=m}^{n} f(i)$$

If n is equal to $m-1$, the above product is defined to be 1. If n is less than $m-1$, the above product is defined as

$$\prod_{i=n+1}^{m-1} \frac{1}{f(i)}$$

If f and F are functions such that for all n,

$$f(n) = \frac{F(n+1)}{F(n)}$$

then $F(n)$ is called the **antiquotient** of $f(n)$. Using telescoping cancellation, it is easy to see that

$$\prod_{i=m}^{n} f(i) = f(m) \cdot f(m+1) \cdots f(n-1) \cdot f(n)$$

$$= \frac{F(n+1)}{F(m)}$$

Thus finding the antiquotient of an expression enables you to determine closed-form **definite products** of the expression with variable upper or lower limits. This is analogous to using antiderivatives to compute definite integrals.

Use the **Calculus Product** command to find the antiquotient or the definite product of an expression. After you select an expression, it prompts you to enter the name of the index variable, and the lower and upper limits. If the limit fields are made blank, the antiquotient is returned; otherwise the definite product is returned.

Alternatively, the antiquotient of u with respect to the variable n can be entered by authoring the form PRODUCT(u, n). For example,

```
PRODUCT(n^2, n)
```

displays using the usual Π (Pi) product notation as

$$\prod_{n}^{n^2}$$

and simplifies to $(n-1)!^2$.

Note that the *upper* case Greek letter for pi (Π) used for products is distinct from the *lower* case Greek letter for pi (π) used to denote the ratio of the circumference to the diameter of a circle (3.14159...).

The definite product of the expression u with respect to n from k to m can be entered by authoring the form PRODUCT(u, n, k, m). For example,

```
PRODUCT(n^2, n, 1, m)
```

displays as

$$\prod_{n=1}^{m} n^2$$

and simplifies to $m!^2$.

To avoid definite products simplifying to expressions involving the FLOOR function (see Section 6.7), ***DERIVE*** assumes variables occurring in the products' upper and lower limits are integers. Therefore, substituting noninteger values for these variables may give incorrect results.

As with antiderivatives and antidifferences, closed-form antiquotients may not exist in terms of the operators and functions known to ***DERIVE***. Even when such an antiquotient exists, there is no known method that is guaranteed to find it.

DERIVE uses several techniques that enable it to find the antiquotient of a large class of expressions. These expressions include monomials, exponentials, products of these, and telescoping products.

Of course, if the upper and lower limits are actual numbers, the product can be computed directly without having to find the antiquotient. For example,

```
PRODUCT(k^2+1, k, 1, 5)
```

simplifies to 44200 even though a closed-form antiquotient does not exist.

If the third argument of PRODUCT is a vector, the index variable ranges over the elements of the vector. For example,

```
PRODUCT(k^2, k, [2, 3, 5, 7, 11])
```

simplifies to 5336100, the product of the squares of the first five prime numbers.

Finally, if given a single argument which is a vector, PRODUCT returns the product of the elements. For example,

```
PRODUCT([2, 3, 5, 7, 11])
```

simplifies to 2310, the product of the first five prime numbers.

Chapter 8. Vectors and Matrices

This chapter describes how vectors and matrices are entered and manipulated using ***DERIVE***'s pre-defined functions and operators. For a demonstration, load the file MATRIX.DMO using a **Transfer Demo** command (see Section 2.9). Skip this chapter if you are not concerned with vectors and matrices.

The file VECTOR.MTH defines functions that perform additional vector and matrix operations (see Section 9.2).

The only limitation on the number of elements in vectors and matrices is the amount of space available for storing their elements. Thus, the maximum number of elements depends strongly on the mathematical complexity of the elements. Many of the matrix operations described in this chapter may dramatically increase the complexity of the resulting elements.

8.1 Entering Vectors and Matrices

A vector is an ordered sequence of elements. Use the **Declare vectoR** command to enter a vector. After requesting the dimension (i.e. the number of elements) of the vector, this command prompts you to enter a value for each of the elements of the vector. If you press **Ctrl-Enter** or **Ctrl-J** instead of **Enter** after specifying the last element, the elements of the resulting vector are simplified.

Alternatively, a vector can be entered by authoring an expression of the form $[x_1, x_2, \ldots, x_n]$ where the x_i are the elements of the vector. For example,

```
[2·x, -5, x^2]
```

is a vector consisting of 3 elements. Note that vectors must be delimited by square brackets [ ], *not* by parentheses ( ) or braces { }.

A matrix is a vector of vectors with each inner (row) vector having the same number of elements. Use the **Declare Matrix** command to enter a matrix. After requesting the dimensions of the matrix, this command prompts you to enter a value for each of the elements of the matrix. If you press **Ctrl-Enter** or **Ctrl-J** instead of **Enter** after specifying the last element, the elements of the resulting matrix are simplified.

Alternatively, a matrix can be entered by authoring a vector of vectors. For example,

```
[[4, a, -5], [a^2, 5, 8·a^2]]
```

displays as the 2 x 3 matrix

$$\begin{bmatrix} 4 & a & -5 \\ a^2 & 3 & 8 \cdot a^2 \end{bmatrix}$$

Note that, unlike vectors, the elements of matrices are *not* separated by commas.

Exercise: Enter and note the difference in the displays of the three-element vector

```
[2·x, -5, x^2]
```

the 1 x 3 matrix

```
[[2·x, -5, x^2]]
```

and the 3 x 1 matrix

```
[[2·x], [-5], [x^2]]
```

Edit a vector or matrix by highlighting it and pressing **F3** to copy it to the author line (see Section 3.3). Alternatively, the **Manage Substitute** command may be useful for replacing one or more elements of a vector or matrix (see Section 4.8).

Exercise: Enter the matrix

$$\begin{bmatrix} 4 & a & -5 \\ a^2 & 3 & 8 \cdot a^2 \end{bmatrix}$$

and use the **Manage Substitute** command to replace all occurrences of a with b in the matrix. Next, use the direction keys (see Section 3.3) to highlight the element a^2 in the original matrix. Finally, use the **Manage Substitute** command to replace all occurrences of a^2 with b^2. Closely compare the resulting matrices.

8.2 Generating Vectors, Matrices, and Tables

Use the **Calculus Vector** command or the VECTOR function to generate a vector of the values of an expression evaluated at a sequence of points. After you select an expression, the **Calculus Vector** command prompts you to enter the name of a variable, the starting value, the ending value, and the step size.

Alternatively, an expression of the form VECTOR(u, k, n) simplifies to a vector of n elements generated by simplifying the expression $u(k)$ with the variable k stepping from 1 through n in steps of size 1. For example,

```
VECTOR(x^2, x, 5)
```

simplifies to [1, 4, 9, 16, 25].

VECTOR(u, k, m, n) simplifies to a vector of $n-m+1$ elements generated by simplifying the expression $u(k)$ with the variable k stepping from m through n in steps of size 1. For example,

```
VECTOR(j!, j, 0, 4)
```

simplifies to [1, 1, 2, 6, 24].

VECTOR(u, k, m, n, s) simplifies to a vector of $(n-m)/s+1$ elements, rounded down, generated by simplifying the expression $u(k)$ with the variable k stepping from m through n in steps of size s. For example,

```
VECTOR(SIN(z), z, 0, π/4, 0.2)
```

approximates to [0, 0.198669, 0.389418, 0.564642].

If the third argument of VECTOR is a vector, the variable ranges over the elements of the vector. For example,

```
VECTOR(k^2, k, [2, 3, 5, 7, 11])
```

simplifies to `[4, 9, 25, 49, 121]`. That is a vector of the squares of the first five prime numbers.

Since a matrix is just a vector of vectors, a nested call on VECTOR can be used to generate a matrix. For example,

```
VECTOR(VECTOR(j + k, k, 1, 4), j, 1, 3)
```

simplifies to the 3 x 4 matrix having elements equal to the sum of their indices:

$$\begin{bmatrix} 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \\ 4 & 5 & 6 & 7 \end{bmatrix}$$

Exercise: Use VECTOR to make a table (i.e. a matrix) of the square and the cube of the integers 1 through 10. Each row of the table should consist of an integer, its square, and its cube.

> **Hint:** The *n*th row of the table can be made by the vector
>
> $$[n, n^2, n^3]$$

An **identity matrix** is a square matrix whose elements are 1 on the main diagonal and 0 off the main diagonal. IDENTITY_MATRIX(*n*) simplifies to an $n \times n$ identity matrix. For example,

```
IDENTITY_MATRIX(3)
```

simplifies to the 3 x 3 identity matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The **dimension** of a vector is the number of elements of the vector. Use DIMENSION to compute the dimension of a vector. Since a matrix is just a vector of vectors, the dimension of a matrix is the number of *rows* of the matrix. For example,

$$\text{DIMENSION}\begin{bmatrix} a & b & c \\ 1 & 2 & 3 \end{bmatrix}$$

simplifies to 2 since the matrix has two rows, and

$$\text{DIMENSION}\left[\text{ELEMENT}\left[\begin{bmatrix} a & b & c \\ 1 & 2 & 3 \end{bmatrix}, 1\right]\right]$$

simplifies to 3 since the first row of the matrix has three columns.

Use ABS to compute the magnitude (length) of a vector. The absolute value of a vector is the square root of the sum of the squares of its elements. For example,

```
ABS [a, b, c]
```

simplifies to

$$\sqrt{(a^2 + b^2 + c^2)}$$

8.3 Vector Manipulation Functions

You can use the direction keys (see Section 3.3) to highlight the elements of a vector. Since a matrix is just a vector of vectors, you can also use these keys to highlight the rows and elements of a matrix. If a row is highlighted, press **Right arrow** to highlight the next row; press **Down arrow** to highlight the first element of the current row.

Once the desired row or element is highlighted, press the **F3** or **F4** function key (see Section 3.3) to insert the highlighted expression on the author line. To insert a column of a matrix on the author line, you can transpose the matrix (see Section 8.5) and then highlight a row of the transposed matrix.

The functions and operators in this section are for extracting, adding, removing, modifying, and testing vector elements.

Use the SUB infix operator to extract the elements of a vector or matrix. On PC compatible computers SUB can be entered by pressing **Alt-V** on the author line and it will display as ↓. In the algebra window subscripts are displayed using standard subscript notation. For example,

```
[a, b, c] SUB 2
```

displays as

$$[a, b, c]_2$$

and simplifies to `b`. Use SUB twice to extract the elements of a matrix. For example,

```
[[2, 3, 5], [7, 1, 4]] SUB 2 SUB 3
```

displays as

$$\begin{bmatrix} 2 & 3 & 5 \\ 7 & 1 & 4 \end{bmatrix}_{2,3}$$

and simplifies to 4. The right operand of SUB can also be a vector of indices. Thus,

```
[[2, 3, 5], [7, 1, 4]] SUB [2, 3]
```

also simplifies to 4.

The ELEMENT function is equivalent to the SUB operator but is entered using functional notation. ELEMENT(v, n) simplifies to the nth element of the vector v. For example,

```
ELEMENT([a, b, c], 2)
```

simplifies to `b`. ELEMENT(m, j, k) simplifies to the element in row j and column k of the matrix m. For example,

```
ELEMENT([[2, 3, 5], [7, 1, 4]], 2, 3)
```

simplifies to 4. Since the SUB operator is displayed using standard subscript notation, we recommend using it rather than the ELEMENT function.

Use the APPEND function to concatenate two or more vectors. APPEND(v_1, v_2, ..., v_n) simplifies to a vector consisting of the elements of the vectors v_1 through v_n. For example,

```
APPEND([a, b], [c, d], [e, f])
```

simplifies to `[a, b, c, d, e, f]`. If A is a matrix, APPEND(A) simplifies to a vector consisting of the elements of A. For example,

```
APPEND([[a, b], [c, d], [e, f]])
```

also simplifies to `[a, b, c, d, e, f]`.

Exercise: Using APPEND, define the function ADJOIN_ELEMENT(u, v) that adjoins expression u to front of vector v. For example, `ADJOIN_ELEMENT(a, [b, c])` should simplify to `[a, b, c]`.

Exercise: Define the function APPEND_COLUMNS(A, B) that appends the columns of matrices A and B. For example,

$$\texttt{APPEND\_COLUMNS}\left[\begin{bmatrix} a & b \\ c & d \end{bmatrix}, \begin{bmatrix} e & f \\ g & h \end{bmatrix}\right]$$

should simplify to

$$\begin{bmatrix} a & b & e & f \\ c & d & g & h \end{bmatrix}$$

> **Hint**: Use the matrix transpose operator (see Section 8.5) before appending the matrices.

Note that ADJOIN_ELEMENTS and APPEND_COLUMNS are defined in the VECTOR.MTH utility file (see Section 9.2).

Use the DELETE_ELEMENT function to delete an element from a vector. DELETE_ELEMENT(v, n) simplifies to a copy of vector v but with the nth element deleted. For example,

```
DELETE_ELEMENT([a, b, c], 2)
```

simplifies to `[a, c]`. Note that if *v* is a matrix, the *n*th row of *v* is deleted.

Exercise: Use DELETE_ELEMENT and the matrix transpose operator (see Section 8.5) to define DELETE_COLUMN(*A*, *k*) that deletes the *k*th column of matrix *A*.

Exercise: Define and test the functions FIRST(*v*) that simplifies to the first element of vector *v* and REST(*v*) that simplifies to all but the first element of vector *v*. For example, FIRST [*a*, *b*, *c*] simplifies to *a*, and REST [*a*, *b*, *c*] simplifies to [*b*, *c*].

Use the INSERT_ELEMENT function to insert an element into a vector. INSERT_ELEMENT(*u*, *v*, *n*) simplifies to a copy of vector *v* but with expression *u* inserted before the *n*th element. For example,

```
INSERT_ELEMENT(d, [a, b, c], 2)
```

simplifies to `[a, d, b, c]`. *n* defaults to 1. If *n* is one more than the dimension of the vector, the new element is added to the end of the vector.

Use the REPLACE_ELEMENT function to replace an element in a vector. REPLACE_ELEMENT(*u*, *v*, *n*) simplifies to a copy of vector *v* but with expression *u* replacing the *n*th element. For example,

```
REPLACE_ELEMENT(d, [a, b, c], 2)
```

simplifies to `[a, d, c]`. *n* defaults to 1.

Use the REVERSE_VECTOR function to reverse the elements of a vector. REVERSE_VECTOR(*v*) simplifies to a copy of vector *v* but with the elements reversed. For example,

```
REVERSE_VECTOR([a, b, c])
```

simplifies to `[c, b, a]`.

Use the SELECT function to find those elements of a vector that satisfy some test criteria. SELECT(*u*, *k*, *v*) simplifies to a vector of those elements *k* of *v* for which *u*(*k*) is true. For example,

```
SELECT(PRIME(k), k, [3, 5, 7, 9, 11])
```

simplifies to `[3, 5, 7, 11]`. SELECT can also be used to test the elements of a sequence. SELECT(*u*, *k*, *m*, *n*, *s*) simplifies to a vector of those *k* for which *u*(*k*) is true with the variable *k* stepping from *m* through *n* in steps of size *s*. *s* defaults to 1. For example,

```
SELECT(PRIME(k), k, 1, 100)
```

simplifies to a vector of all the prime numbers less than 100.

8.4 Vector Operations

Since a matrix is just a vector of vectors, the vector operations described in this section also apply to matrices.

Use the + (plus) operator to add the corresponding elements of two vectors. Use the – operator to subtract the corresponding elements of two vectors. In either case, the vectors must have the same dimension. For example,

$$[2 \cdot x, -5, x^2] + [-x, 8, 2 \cdot x]$$

simplifies to

$$[x, 3, x^2 + 2 \cdot x]$$

Use the * (times) operator or implicit multiplication (see Section 3.1) to multiply each element of a vector by a scalar. Use the / operator to divide each element of a vector by a scalar. For example,

$$3\ [2 \cdot x, -5, x^2]$$

simplifies to

$$[6 \cdot x, -15, 3 \cdot x^2]$$

Use the . (period) infix operator to compute the **dot product** (also called the **inner product**) of vectors and matrices. To prevent a period used for the dot product operator from being interpreted as a decimal point, use a space to separate the period from numbers that immediately precede or follow it.

The * operator or implicit multiplication can also be used to compute dot products provided the operands are vectors or have variables that have been declared nonscalar (see Section 4.10).

The dot product of two vectors (also called the **scalar product**) is the sum of the products of corresponding elements of the vectors. The dimensions of the vectors must be the same. Note that the result is a scalar. For example,

$$[2, a, 5] \cdot [2 \cdot a, 3, -1]$$

simplifies to

$$7 \cdot a - 5$$

The dot product of a matrix and a vector is the vector whose elements are the dot product of the vector and the *rows* of the matrix. The dimension of the vector must be the same as the dimension of the rows of the matrix. For example,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot [2,\ 3]$$

simplifies to

$$[2 \cdot a + 3 \cdot b,\ 2 \cdot c + 3 \cdot d]$$

The dot product of a vector and a matrix is the vector whose elements are the dot product of the vector and the *columns* of the matrix. The dimension of the vector must be the same as the dimension of the matrix. For example,

$$[2,\ 3] \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

simplifies to

$$[2 \cdot a + 3 \cdot c,\ 2 \cdot b + 3 \cdot d]$$

The dot product of two matrices is the matrix whose elements are the dot product of the *rows* of the first matrix and the *columns* of the second matrix. The number of columns of the first matrix must equal the number of rows of the second matrix. For example,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

simplifies to

$$\begin{bmatrix} 2 \cdot a + 3 \cdot b \\ 2 \cdot c + 3 \cdot d \end{bmatrix}$$

As another, more general, example,

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \cdot \begin{bmatrix} a & b & 1 \\ c & d & 1 \end{bmatrix}$$

simplifies to

$$\begin{bmatrix} 2 \cdot a + 3 \cdot c & 2 \cdot b + 3 \cdot d & 5 \\ 4 \cdot a + 5 \cdot c & 4 \cdot b + 5 \cdot d & 9 \end{bmatrix}$$

Use CROSS to compute the **cross product** of two vectors. Both vectors must have three elements or both must have two elements. For example,

```
CROSS([1, 2, 3], [a, b, c])
```

simplifies to

```
[2·c - 3·b, 3·a - c, b - 2·a]
```

and

```
CROSS([1, 2], [a, b])
```

simplifies to

```
b - 2·a
```

8.5 Matrix Operations

The functions and operators described in this section only apply to matrices.

The **transpose** of a matrix is the matrix resulting from interchanging its rows and columns. The transpose of a scalar or a vector of scalars is the original scalar or vector of scalars.

Use the ` (back-accent) postfix operator to compute the transpose of a matrix. (On PC-9801 compatible computers, the ¥ yen character is the matrix transpose operator.) For example,

```
[[a, b, c], [1, 2, 3]]`
```

displays as

$$\begin{bmatrix} a & b & c \\ 1 & 2 & 3 \end{bmatrix}^{\backprime}$$

and simplifies to

$$\begin{bmatrix} a & 1 \\ b & 2 \\ c & 3 \end{bmatrix}$$

A **square matrix** is a matrix with the same number of rows as columns. Use DET to compute the **determinant** of a square matrix. For example,

```
DET([[2, 3], [a, b]])
```

displays as

$$\mathrm{DET}\begin{bmatrix} 2 & 3 \\ a & b \end{bmatrix}$$

and simplifies to

```
2·b − 3·a
```

The **trace** of a square matrix is the sum of its diagonal elements. Use TRACE to compute the trace of a square matrix. For example,

$$\mathrm{TRACE}\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

simplifies to 15.

Use the ^ operator to raise a square matrix to an integer power. For example,

```
[[2, 3], [a, b]] ^ 2
```

displays as

$$\begin{bmatrix} 2 & 3 \\ a & b \end{bmatrix}^2$$

and simplifies to

$$\begin{bmatrix} 3 \cdot a + 4 & 3 \cdot b + 6 \\ a \cdot (b + 2) & 3 \cdot a + b^2 \end{bmatrix}$$

Exercise: Find the inverse of the square matrix

$$\begin{bmatrix} 2 & 3 \\ a & b \end{bmatrix}$$

by raising it to the −1 power. Then take the inverse of the resulting matrix. Note that the final result is equal to the original matrix and that *no* roundoff errors were introduced.

A **singular matrix** is a square matrix whose determinant is 0. A singular matrix has no inverse. For example,

$$\begin{bmatrix} 2 & 3 \\ 4 \cdot a & 6 \cdot a \end{bmatrix}^{-1}$$

does not simplify further.

Inverting a matrix can produce dramatically more complicated elements than the original matrix. The complexity of the inverse of an $n \times n$ matrix with symbolic elements often grows combinatorially with n. Thus the inverse of symbolic matrices having more than 25 nonzero elements may exhaust the 640 kilobytes of memory available to ***DERIVE***. Given enough extended memory, the only limit for ***DERIVE XM*** is the amount of time required to compute the inverse.

Even for numeric matrices, the elements of an inverse computed in exact or mixed mode can have dramatically more digits than the given elements. Thus it may be possible to enter a 50 x 50 matrix whose elements are all small magnitude integers, and perform a modest operation such as computing the trace of the matrix.

A system of linear equations can be represented as a vector of constants that is equal to the dot product of a coefficient matrix and a vector of unknowns. Provided the matrix is nonsingular, you can use the matrix inverse and dot product operations to solve this matrix equation. For example, the system

$$\begin{aligned} 5\cdot x + 3\cdot y - 7\cdot z &= 4 \\ 2\cdot x - 8\cdot y + z &= 6 \\ -x + 9\cdot y + 4\cdot z &= 5 \end{aligned}$$

is equivalent to the matrix equation

$$\begin{bmatrix} 5 & 3 & -7 \\ 2 & -8 & 1 \\ -1 & 9 & 4 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 5 \end{bmatrix}$$

Multiplying both sides of the equation by the inverse of the coefficient matrix yields

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 & 3 & -7 \\ 2 & -8 & 1 \\ -1 & 9 & 4 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 4 \\ 6 \\ 5 \end{bmatrix}$$

which approximates to

$$\begin{bmatrix} x = 2.91059 \\ y = 0.175496 \\ z = 1.58278 \end{bmatrix}$$

Note that the exact rational solution can be found by simplifying instead of approximating the equation.

The above method is only able to solve systems of linear equations that are *non*singular. Also since it requires the explicit computation of the inverse of a matrix, it is both space and time inefficient. In contrast, the **soLve** command (see Section 4.15) and the function ROW_REDUCE (see the next section) can solve larger as well as singular systems of linear equations, so long as they are consistent.

Exercise: Solve the above matrix equation using the **soLve** command in both approximate and exact mode.

8.6 Row Echelon Form

Another method for solving systems of linear equations involves reducing a matrix to row echelon form. In addition to being able to solve both singular and nonsingular systems, this method can simultaneously determine solutions to more than one set of constants on the right side of the equations.

Row echelon form is sometimes called reduced row echelon form or Hermite normal form. A matrix is in row echelon form if

- the first nonzero element in every row is 1,
- the first nonzero element in every row occurs to the right of the first nonzero element in the row above it, and
- all the elements above the first nonzero element in a row are 0.

Use ROW_REDUCE to compute the row echelon form of a matrix. The only matrix operations used are multiplying rows by scalars and adding rows. These **elementary row operations** do not change the solution set. Also the rank of a matrix is equal to the number of nonzero rows in the row echelon form of the matrix.

If ROW_REDUCE is given two matrices, it adjoins them and returns the row echelon form of this augmented matrix. Thus, to solve the matrix equation

$$A \cdot X = B$$

where A and B are matrices, enter and simplify (or approximate)

$$\text{ROW\_REDUCE}(A,\ B)$$

If A is nonsingular, the result is a matrix in row echelon form: the identity matrix adjoined to the solution matrix. For example,

$$\text{ROW\_REDUCE}\left[\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix},\ \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix}\right]$$

simplifies to

$$\begin{bmatrix} 1 & 0 & -1 & -2 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

and the solution is

$$\begin{bmatrix} -1 & -2 \\ 2 & 3 \end{bmatrix}$$

Exercise: Solve the matrix equation example given in the previous section by approximating the expression

$$\texttt{ROW\_REDUCE}\left[\begin{bmatrix} 5 & 3 & -7 \\ 2 & -8 & 1 \\ -1 & 9 & 4 \end{bmatrix}, \begin{bmatrix} 4 \\ 6 \\ 5 \end{bmatrix}\right]$$

If A is singular, rows having 0 on the main diagonal are consistent only with augmented columns having 0 in the same row, in which case the corresponding element of X is arbitrary. For example, since

$$\texttt{ROW\_REDUCE}\left[\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}\right]$$

simplifies to

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

the right column is inconsistent, but the column preceding it is consistent.

8.7 Eigenvalues

The characteristic polynomial of a square matrix is the determinant of the difference of the matrix and a variable times the identity matrix. If A is a square matrix and v is a variable, CHARPOLY(A, v) simplifies to the characteristic polynomial of A in terms of v. For example,

```
CHARPOLY([[2, 3], [a, b]], z)
```

simplifies to

$$z^2 - z \cdot (b + 2) - 3 \cdot a + 2 \cdot b$$

The default value of the variable used by CHARPOLY is w.

The eigenvalues of a square matrix are the zeros of its characteristic polynomial. If A is a square matrix and v is a variable, EIGENVALUES(A, v) simplifies to the eigenvalues of A in terms of v. For example,

```
EIGENVALUES([[2, 3], [0, b]], z)
```

simplifies to

```
[z = 2, z = b]
```

The default value of the variable used by EIGENVALUES is w.

The degree of the characteristic polynomial equation that EIGENVALUES must solve is equal to the dimension of the matrix. Since characteristic polynomials rarely factor exactly over the rational numbers, exact results usually require the cubic formula for 3 x 3 matrices or the quartic formula for 4 x 4 matrices. Exact results are usually impossible for matrices larger than 4 x 4. Moreover, the quartic formula generates such bulky results that the exact eigenvalues of 4 x 4 matrices often exhaust memory. Thus, exact eigenvalues are rarely attainable for matrices larger than 3 x 3.

The characteristic polynomial of a numerical matrix is univariate. Thus real-valued eigenvalues can be approximated by plotting the characteristic polynomial to find the zeros and then solving it using the **soLve** command in approximate mode (see Section 4.14). Note that the eigenvalues of symmetric and Hermitian matrices are all real. For matrices with complex eigenvalues, the utility functions NEWTONS or FIXED_POINT (see Section 9.1) can be used to find the complex zeros of their characteristic polynomials.

The file VECTOR.MTH defines functions for computing eigenvectors (see Section 9.2).

8.8 Vector and Matrix Algebra

DERIVE automatically uses the rules of vector and matrix algebra to simplify and normalize expressions involving vectors and matrices. If *a*, *b*, and *c* are declared nonscalar (see Section 4.10), then:

$$a`` \rightarrow a$$

$$(a^{-1})` \rightarrow a`^{-1}$$

$$(a + b)` \rightarrow a` + b`$$

$$(a \cdot b)` \rightarrow b` \cdot a`$$

$$a \cdot (b + c) \rightarrow a \cdot b + a \cdot c$$

$$(b + c) \cdot a \rightarrow b \cdot a + c \cdot a$$

$$(a \cdot b) \cdot c \rightarrow a \cdot (b \cdot c)$$

$$(a \cdot b)^{-1} \rightarrow b^{-1} \cdot a^{-1}$$

$$\text{DET}(a^{-1}) \rightarrow \frac{1}{\text{DET}(a)}$$

8.9 Differential Vector Calculus

DERIVE can do differential vector calculus in any orthogonal curvilinear coordinate system. For example, you can compute gradients, divergences, and curls in Cartesian, polar-cylindrical, or spherical coordinates. The default coordinate system is three-dimensional rectangular-Cartesian with x, y, and z used as the names of the coordinate variables.

Use GRAD to compute the **gradient** of an expression. For example,

$$\text{GRAD}(x \cdot y^2 \cdot z^3)$$

simplifies to

$$[\, y^2 \cdot z^3,\ 2 \cdot x \cdot y \cdot z^3,\ 3 \cdot x \cdot y^2 \cdot z^2 \,]$$

To compute the general form of the gradient in the Cartesian coordinate system, define F to be an arbitrary function (see Section 4.12). Then

$$\text{GRAD}(F(x,\ y,\ z))$$

simplifies to

$$\left[\frac{d}{dx} F(x,\ y,\ z),\ \frac{d}{dy} F(x,\ y,\ z),\ \frac{d}{dz} F(x,\ y,\ z)\right]$$

To specify Cartesian coordinates with other coordinate variables, call GRAD with a vector of coordinate variables as the second argument. For example,

$$\text{GRAD}(c \cdot w + x^2 + y^3 + z^4,\ [w,\ x,\ y,\ z])$$

simplifies to the four-element vector

$$[\, c,\ 2 \cdot x,\ 3 \cdot y^2,\ 4 \cdot z^3 \,]$$

For other orthogonal coordinate systems, provide a second argument that is a **coordinate geometry matrix** of the form

$$\begin{bmatrix} x_1 & x_2 & & x_n \\ h_1 & h_2 & & h_n \end{bmatrix}$$

The first row consists of the coordinate variables associated with successive elements of vectors. The second row consists of the scale factors such that the squared infinitesimal length is given by

$$ds^2 = (h_1 \cdot dx_1)^2 + (h_2 \cdot dx_2)^2 + \ldots + (h_n \cdot dx_n)^2$$

The file VECTOR.MTH (see Section 9.4) assigns polar-cylindrical and spherical coordinate geometry matrices to the variables cylindrical and spherical respectively as follows:

```
cylindrical := [ r  θ  z ]
               [ 1  r  1 ]
```

and

```
spherical := [ r      θ      Φ ]
             [ 1  r·SIN(Φ)   r ]
```

Thus, if VECTOR.MTH is loaded (see Section 2.9),

```
GRAD(r·SIN(θ)·COS(Φ), spherical)
```

simplifies to

```
[COS(Φ)·SIN(θ), COT(Φ)·COS(θ), - SIN(Φ)·SIN(θ)]
```

For a right-hand coordinate system having the positive z-axis associated with the north pole, this definition of spherical coordinates measures colatitude Φ in radians south from the north pole and measures colongitude Θ in radians east from the positive *x*-axis. Be aware that some references define spherical coordinates differently, such as interchanging the definitions of Θ and Φ.

All the differential vector functions take an optional second argument that is a Cartesian coordinate vector or a coordinate geometry matrix.

Use DIV to compute the **divergence** of a vector. For example,

```
DIV([y^2·z^3, 2·x·y·z^3, 3·x·y^2·z^2])
```

simplifies to

```
x·(6·y^2·z + 2·z^3)
```

Use LAPLACIAN to compute the divergence of the gradient of an expression. For example,

```
LAPLACIAN(x·y^2·z^3)
```

simplifies to

```
x·(6·y^2·z + 2·z^3)
```

Use CURL to compute the **curl** of a vector having two or three elements. For example,

```
CURL([y^2, 2·x·z, 0])
```

simplifies to

```
[-2·x, 0, 2·z - 2·y]
```

The curl of a planar vector field is a vector perpendicular to that plane. Thus, the curl of a two element vector is a three element vector whose first two elements are zero. In this case ***DERIVE*** follows the more common convention of just returning the third element as a scalar. For example,

```
CURL([v^2, u], [[u, v], [1, 1]])
```

simplifies to

```
1 - 2·v
```

8.10 Integral Vector Calculus

DERIVE can do integral vector calculus in any orthogonal curvilinear coordinate system. For example, you can compute scalar or vector potentials in Cartesian, polar-cylindrical or spherical coordinates.

Use POTENTIAL to compute a **scalar potential** of a vector. For example,

```
POTENTIAL([y^2·z^3, 2·x·y·z^3, 3·x·y^2·z^2])
```

simplifies to

$$x \cdot y^2 \cdot z^3$$

As with antiderivatives, scalar potentials are unique only to within a constant. Thus, POTENTIAL may give a different result than one derived manually. Such additive constants can be well disguised — particularly in expressions involving logarithms or inverse trigonometric functions. However, the gradient of a potential should be equivalent to the original vector.

Not all vectors have an associated scalar potential. POTENTIAL merely computes a certain line integral involving the given vector. If there is no potential, then the gradient of the resulting scalar will not be equivalent to the original vector. It is your responsibility to check this. In the case of a vector of two or three elements, it may be easier to see if the CURL of the given vector is equivalent to zero, as is necessary and sufficient for the existence of a scalar potential.

POTENTIAL takes an optional second argument that is a vector specifying the starting coordinates for the line integrals. This argument defaults to a vector of zeros. An inappropriate choice may lead to an infinite or unknown potential. Try other values if this happens. What you want is starting coordinates at which the given gradient is simple and finite. $\pi/2$ and other simple rational multiples of π are often good alternatives for angular starting coordinates. ∞ is often a good starting nonangular coordinate when 0 is not. 1 is often a good starting coordinate if the given gradient involves logarithms of that coordinate.

POTENTIAL also takes an optional third argument that is a Cartesian coordinate vector or a coordinate geometry matrix (see Section 8.9). The default corresponds to three-dimensional Cartesian coordinates using successive variables x, y, and z.

Use VECTOR_POTENTIAL to compute a **vector potential** of a three-element vector. For example,

```
VECTOR_POTENTIAL([x, 0, y - z])
```

simplifies to

$$\left[-\frac{y^2}{2}, -x \cdot z, 0\right]$$

As with POTENTIAL, VECTOR_POTENTIAL takes optional second and third arguments for specifying the starting coordinates and a coordinate geometry matrix.

Vector potentials are unique only to within an arbitrary gradient. Consequently, two equally valid vector potentials may be quite different. However, the curls of both vector potentials should be equivalent to the original vector.

Not all vectors have an associated vector potential. VECTOR_POTENTIAL merely computes a certain line integral involving the given vector. If there is no vector potential, the curl of the resulting vector will not be equivalent to the original vector. It is your responsibility to check this. Alternatively, it may be easier to see if the DIV of the given vector is zero, as is necessary and sufficient for the existence of a vector potential.

Chapter 9. Utility Files

Utility files are collections of function definitions and variable assignments. Chapter 4 describes the utility files ENGLISH.MTH, METRIC.MTH, and PHYSICAL.MTH. This chapter describes the more advanced utility files.

Utility files can be loaded using either the **Transfer Load Derive** command, the **Transfer Merge** command, or the **Transfer Load Utility** command (see Section 2.9). The first two commands have the advantage of displaying the function names and their arguments. The **Transfer Load Utility** command has the advantage of loading utility files faster and liberating some screen and memory space by not displaying function definitions.

Utility files can be loaded in either character input mode or word input mode (see Section 4.1). However, if you need to refer to the multi-character variable names used in a utility file, word input mode must be selected using an **Options Input** command *before* entering the variable name.

If you often use only a particular subset of the definitions in a utility file, it saves time and space to copy just those definitions to a separate file. You can do this by loading the file into an empty algebra window, removing all but the desired definitions, and then using the **Transfer Save** command (see Section 2.8) to save the remaining definitions under a *new* file name. When doing this, check the definition of each desired function to see if it requires other functions in the file, which in turn may require additional functions in the file. If you know how to use a text editor that provides an ASCII file option, it may be more convenient for marking portions of a utility file and writing those portions to new files. For readability, many of the function names are unabbreviated phrases. If you use them often, you may want to shorten the names in such new files.

The default values of trailing arguments that are omitted are the current values of the parameters used in the definitions (see Section 4.12). Most of the definitions in these files use commonly occurring names such as x for an abscissa, y for an ordinate, and r for a radius. Consequently, you can often omit one or more trailing arguments if their values are the current values of the parameter names. This includes the most common case of parameters whose values are merely their own names.

The files include some auxiliary functions that are not described here. They are not intended for you to use directly and, consequently, you do not need to know how to use them. Nonetheless, you may learn some good programming techniques by studying them and the functions that use them.

9.1 SOLVE.MTH — Solving Nonlinear Systems and Complex Solutions

The **soLve** command and the pre-defined SOLVE function can determine real or complex solutions to simultaneous linear algebraic equations (see Section 4.15). For a single nonlinear equation, they can also determine some exact real or complex solutions or approximate real solutions (see Section 4.14). To supplement these capabilities, the file SOLVE.MTH defines functions that can be used to find approximate solutions (real or complex) to systems of one or more simultaneous algebraic equations by iterative methods.

To solve a system of m algebraic equations in variables x_1 through x_m, first rearrange the equations into the form $u_i=0$ for $i=1$ through m. Next make a vector u of the form $[u_1, u_2, \ldots, u_m]$. Then if ***DERIVE*** can generate a matrix of symbolic partial derivatives of each u_i with respect to each x_i without exhausting memory, the function NEWTONS can be used to solve the system of equations as follows:

NEWTONS(u, x, x_0, n) approximates to an $n+1$ x m matrix. x is the vector of variables $[x_1, x_2, \ldots, x_m]$, and x_0 is the vector of initial guesses for the corresponding variables of x. The first row of the resulting matrix is the initial guess, and subsequent rows are successive values of the variables computed using Newton's method (see Section 10.1). Note that it is almost always appropriate to approximate rather than simplify when using NEWTONS.

For example, to solve the system of nonlinear equations

$$\hat{e}^{x \cdot y} = x + 2 \cdot y$$

$$\text{ATAN}(x \cdot y) = 2 \cdot x + y$$

first use the **Transfer Load Utility** command (see Section 2.9) to load the file SOLVE.MTH if not already loaded. Then enter and approximate the expression

```
NEWTONS([ê^(x y)-x-2y, ATAN(x y)-2x-y], [x, y], [-1, -1], 6)
```

This will result in a 7 x 2 matrix of numbers whose last row indicates that $x=-0.424667$ and $y=0.599887$ is a solution to the equations. This solution can be verified by substitution into the original equations.

If n is omitted in a call on NEWTONS, iteration continues until a row is identical to one of the earlier rows. Note that numbers that look identical in scientific or decimal notation may actually correspond to different internal rational numbers.

If you are interested in solutions containing complex numbers, you can use such numbers in the components of your initial guess. When you find a complex solution, its conjugate is often a good guess for another solution since complex solutions often occur in conjugate pairs.

Even if there is a solution, Newton's method may not converge to it unless your initial guess is sufficiently close. Also, divergence may generate large magnitude numbers that exhaust memory and/or take a long time to abort. Thus, initially it is best to call NEWTONS with n equal to a small positive integer value. Then by examining the last few rows of the resulting matrix, you can determine whether Newton's method is converging. If convergence does not begin within ten iterations or so, it is usually best to try a different initial guess rather than increasing n.

If s is the last row of the resulting matrix, you can approximate

$$\lim_{x \to s} u$$

to determine if the residual is acceptably small. If s has converged to within roundoff error at the current precision and you want more accuracy, increase the precision by a few digits and use s as your initial guess. This is usually more efficient than doing the whole calculation at the higher precision.

If Newton's method is taking an unexpectedly long time, the iteration may be diverging. Press **Esc** to abort the computation, and retry using a smaller n or a better initial guess. Remember to use the **approX** command rather than the **Simplify** command when using NEWTONS.

NEWTONS only finds one solution at a time. To find additional solutions, you can use a different initial guess. However, NEWTONS may still converge to the solution you already know. To prevent this, the original system can be reduced to a deflated system of equations. If $x=[s_1, s_2, \ldots, s_m]$ is a solution to the system represented by the vector u, the vector for the deflated system is

$$\frac{u}{(x_1-s_1)(x_2-s_2)\ldots(x_m-s_m)}$$

If you find a solution to the deflated system, you can apply NEWTONS to the original system using this solution as an initial guess. This eliminates roundoff error caused by deflation and gives a more accurate, purified solution.

The difficulty of finding sufficiently good initial guesses increases sharply with the number of variables. Consequently, it is usually wise to eliminate as many variables as possible before resorting to NEWTONS (see Section 4.15).

To solve a single equation using Newton's method, the function NEWTON defined in Section 10.1 can be used instead of NEWTONS, since it is easier to use.

Exercise: 3D plot the sum of the absolute values of the left sides of the equations

$$3 \cdot x \cdot y - 2 \cdot x^2 + 4 \cdot \mathrm{SIN}(y) + 6 = 0$$

$$3 \cdot x^2 - 2 \cdot x \cdot y^2 + 3 \cdot \cos(x) + 4 = 0$$

to obtain guesses for where they are simultaneously 0. Then use NEWTONS to approximate all the real solutions.

Exercise: Use NEWTONS to determine the six complex zeros of the polynomial

$$z^6 + z^4 - 12 \cdot z^3 + 28 \cdot z^2 - 48 \cdot z + 161$$

For appropriate initial guesses, substitute $x+\hat{\imath}y$ for z in the polynomial, giving a polynomial p. Then simplify the square of p's magnitude

$$\mathrm{RE}(p)^2 + \mathrm{IM}(p)^2$$

and 3D plot the minimum of the result and 10. By counting grid lines, you can roughly determine the six values of z where the original polynomial is 0. Finally, use NEWTONS with nine iterations on these initial guesses to find the six complex zeros of the polynomial.

Another way to solve a system of equations approximately is to convert the system to a vector fixed-point form $x=g$, with $g=[g_1, g_2, \ldots, g_m]$, where expressions g_1 through g_m may each depend on any or all of the variables in $x=[x_1, x_2, \ldots, x_m]$. You can then substitute the initial guesses for x_1 through x_m into g to compute updated guesses for x. You can continue to substitute the most recent set of components into g to compute a more recent set of guess components.

If there is a solution, the magnitudes of the partial derivatives of g_1 through g_m are small in the neighborhood of the solution, and the initial guess is close to the solution, then fixed-point iteration may converge to the solution and the convergence may occur in an acceptable amount of time. To compensate for these discouraging provisos, fixed-point iteration does not require differentiability and does not require space to store derivatives, unlike Newton's method.

FIXED_POINT(g, x, x_0, n) approximates to an $n+1$ x m matrix. g is the vector of expressions $[g_1, g_2, \ldots, g_m]$. The other arguments, the returned result, and the usage suggestions are similar to those described for NEWTONS. For example, use complex numbers in the components of your initial guess if you are interested in complex solutions.

However, FIXED_POINT may diverge no matter how close the initial guess is to an exact solution: The most important determinants of success are choosing a close enough initial guess and choosing a particular fixed-point form for which g_1 through g_m have small magnitude partial derivatives near the guess: Ideally each equation has a portion that is invertible with respect to a distinct variable, and the magnitude of

the partial derivative of this portion with respect to that variable is dominant near your guess. You can then solve each equation for its dominant variable x_k in its dominant portion in terms of the other portions to obtain the form $x_k = g_k$. The same variable may occur on the right side too, but weakly.

Exercise: Use FIXED_POINT to find all real solutions of the system of equations

$$x - 1 = \frac{1}{1 + (x + y)^4}$$

$$y - 3 = \frac{1}{1 + (x - y)^4}$$

You can use a 3D plot to suggest initial guesses, as described for previous exercises.

Even if an equation cannot be solved exactly and extra parameters make it impossible to solve numerically, it may still be possible to find a truncated Taylor series solution. The series solution can extend an exact or approximate solution at a particular point into a useful neighborhood around the point. Of course, it is necessary that a truncated series solution exists at the point.

TAYLOR_SOLVE (u, x, y, x_0, y_0, n) simplifies to an nth degree truncated Taylor series solution $y(x)$ of the equation $u(x, y)=0$, given $u(x_0, y_0)=0$. For example, since $y_0 = 0$ at $x_0 = 0$ is a solution of the equation $\sin(y) + y + x\hat{e}^x = 0$,

```
TAYLOR_SOLVE(SIN(y) + y + x·ê^x, x, y, 0, 0, 3)
```

simplifies to

$$-\frac{25 \cdot x^3}{96} - \frac{x^2}{2} - \frac{x}{2}$$

This is the 3th degree truncated Taylor series solution of the equation expanded about the point $y_0 = 0$ at $x_0 = 0$. Note that it is impossible to solve this transcendental equation exactly for y. Also the extra parameter x makes it impossible to solve the equation numerically.

TAYLOR_INVERSE (u, x, y, x_0, n) simplifies to an nth degree truncated Taylor series expansion of the inverse of the function defined by $y=u(x)$, expanded about $y_0=u(x_0)$. This is useful when $u(x)$ cannot be inverted exactly in closed form. For example, TAYLOR_INVERSE ($x\hat{e}^x$, x, y, 0, 3) simplifies to

$$\frac{3 \cdot y^3}{2} - y^2 + y$$

9.2 VECTOR.MTH — Additional Vector and Matrix Functions

The file VECTOR.MTH defines functions that automate some of the operations commonly performed on vectors and matrices. Because matrices are just vectors of vectors, any function that operates on the elements of a vector operates similarly on the *rows* of a matrix. Moreover, by transposing a matrix, then applying a function that operates on the elements of a vector, and finally transposing the result, you can perform the same operation on the *columns* of a matrix.

i_ simplifies to the vector [1, 0, 0].

j_ simplifies to the vector [0, 1, 0].

k_ simplifies to the vector [0, 0, 1].

These three assigned variables represent the unit vectors i, j, and k for the Cartesian coordinate system. The underscore in their names is meant to suggest a vector and to distinguish them from the user variables i, j, and k. Any Cartesian coordinate vector can be entered as a linear combination of unit vectors.

Exercise: Simplify the expression

```
a·i_ + 3·j_ - k_
```

OUTER(v, w) simplifies to the outer product of vectors v and w. For example,

```
OUTER([a, b], [2, 3, 4])
```

simplifies to

$$\begin{bmatrix} 2\cdot a & 3\cdot a & 4\cdot a \\ 2\cdot b & 3\cdot b & 4\cdot b \end{bmatrix}$$

KRONECKER(i, j) simplifies to 1 if integer i equals integer j, otherwise it simplifies to 0. The Kronecker delta function of i and j is often written as $\delta_{i,j}$. KRONECKER is a useful utility function for implementing various vector and matrix operations.

The next several functions perform various operations on vectors and matrices:

ADJOIN_ELEMENT(u, v) adjoins expression u to the front of vector v. For example, ADJOIN_ELEMENT(a, [b, c]) simplifies to [a, b, c].

APPEND_COLUMNS(A, B) appends the columns of matrices A and B. For example,

$$\texttt{APPEND\_COLUMNS}\left[\begin{bmatrix} a & b \\ c & d \end{bmatrix}, \begin{bmatrix} e & f \\ g & h \end{bmatrix}\right]$$

simplifies to

$$\begin{bmatrix} a & b & e & f \\ c & d & g & h \end{bmatrix}$$

MINOR(A, i, j) simplifies to a copy of matrix A in which row i and column j are omitted.

SWAP_ELEMENTS(v, i, j) simplifies to a copy of vector v with elements i and j interchanged. For example, SWAP_ELEMENTS([a, b, c], 1, 2) simplifies to [b, a, c]. Note that SWAP_ELEMENTS can be used to interchange the rows of a matrix.

SCALE_ELEMENT(v, i, s) simplifies to a copy of vector v in which element i is multiplied by expression s. For example, SCALE_ELEMENT([a, b, c], 2, 3) simplifies to [a, 3 b, c]. Note that SCALE_ELEMENT can be used to multiply the rows of a matrix by scalars.

SUBTRACT_ELEMENTS(v, i, j, s) simplifies to a copy of vector v in which element i is replaced with element i minus s times element j. s defaults to 1. SUBTRACT_ELEMENTS makes it possible to reduce a matrix to row echelon form using Gaussian or Gauss-Jordan elimination on a matrix one step at a time. This is useful for educational purposes or because you can make a better choice of pivots than does ROW_REDUCE (see Section 8.6).

Exercise: Assign the variable A the matrix value as follows:

$$\texttt{a} := \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 6 & 5 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

Then reduce A to row echelon form directly using ROW_REDUCE (see Section 8.6), and then one row at a time using SUBTRACT_ELEMENTS.

> **Hint**: The first step is to make the assignment
> `a := SUBTRACT_ELEMENTS(a, 2, 1, 5)`

FORCE0(A, i, j, p) simplifies to a copy of matrix A in which element i, j is forced to 0 by subtracting an appropriate multiple of pivot row p from row i. FORCE0 combines scaling and subtraction to force an element of a matrix to 0 in a single step.

Exercise: Reduce the matrix in the previous exercise to row echelon form using FORCE0.

PIVOT(A, i, j) simplifies to a copy of matrix A in which column j below row i is forced to 0 by subtracting appropriate multiples of row i.

Exercise: Reduce the matrix in the previous exercise to row echelon form using PIVOT.

When a direct computation exhausts memory, it is often possible to compute a result in pieces. For really large problems, these partial results can be written to separate files using the **Transfer Save** command and then removed from the algebra window before computing the next partial result. The parallelism inherent in many matrix operations makes this *piecewise* approach especially applicable to such operations. The following two functions facilitate this approach:

MATPROD(A, B, i, j) simplifies to the row i column j element of the dot product of matrix A and matrix B. The elements of the product of two matrices with nonnumeric elements are often much bulkier than the elements of the original matrices. MATPROD allows you to compute the matrix product one *element* at a time.

An alternative intermediate approach is to compute the matrix product one *row* at a time by simplifying expressions of the form ELEMENT(A, i) · B, where i goes from 1 to the dimension of A.

Exercise: Assign the variables A and B matrix values as follows:

$$a := \begin{bmatrix} x & 1 \\ 3 & 2 \cdot x \end{bmatrix} \qquad \text{and} \qquad b := \begin{bmatrix} -1 & x \\ 3 & x \end{bmatrix}$$

Then compute the dot product of A and B directly using the dot product operator (see Section 8.4), one row at a time using the above approach, and one element at a time using MATPROD.

COFACTOR(A, i, j) simplifies to the numerator of the row j column i element of the inverse of square matrix A. The corresponding element of the inverse is this numerator divided by DET(A). Element growth is severe when computing inverses of matrices having nonnumeric elements. COFACTOR allows you to compute the matrix inverse one *element* at a time.

An alternative intermediate approach is to compute the matrix inverse one *column* at a time by simplifying expressions of the form ROW_REDUCE(A, v), where v is a column of the identity matrix for dimension A.

Exercise: Using the matrix value assigned to the variable A in the previous exercise, compute the inverse of A directly using the exponentiation operator (see Section 8.5), one column at a time using ROW_REDUCE, and one element at a time using COFACTOR.

COFACTOR can also be used to compute the determinant of a square matrix A, if simplifying DET(A) exhausts memory. If A is an n x n matrix, the determinant of A is the sum of expressions of the form ELEMENT(A, 1, i)·COFACTOR(A, 1, i), where i goes from 1 to n. The determinant can also be computed by similarly expanding about any row or column of the matrix.

Exercise: Using the matrix value assigned to the variable A in the previous exercise, compute the determinant of A directly using DET (see Section 8.5), and one term at a time using COFACTOR.

ADJOINT(A) simplifies to the adjoint or adjugate of square matrix A. The adjoint of A is the transpose of the matrix of cofactors of A. For example,

$$\text{ADJOINT}\begin{bmatrix} 2 & x \\ 1 & 3 \end{bmatrix}$$

simplifies to

$$\begin{bmatrix} 3 & -x \\ -1 & 2 \end{bmatrix}$$

RANK(A) simplifies to the rank of matrix A. The rank of a matrix is equal to the number of nonzero rows in the row echelon form of the matrix (see Section 8.6). For example,

$$\text{RANK}\begin{bmatrix} 2 & 3 & 5 \\ 4 & 6 & 10 \\ 1 & 2 & 3 \end{bmatrix}$$

simplifies to 2, since the second row is a multiple of the first. Note that in approximate mode, RANK may return erroneous results due to roundoff errors.

If μ is an exact eigenvalue of the square matrix A, the corresponding eigenvector (characteristic vector) is the nonzero vector x that satisfies the singular homogeneous system $(A - \mu \cdot I)\, x = 0$. The eigenvector generally contains m arbitrary parameters, where m is the multiplicity of the eigenvalue. The pre-defined function EIGENVALUES can be used to compute the exact eigenvalues of a matrix (see Section 8.7).

EXACT_EIGENVECTOR(A, μ) simplifies to the eigenvector of matrix A corresponding to exact eigenvalue μ. If μ is only an approximate eigenvalue, the matrix $A - \mu \cdot I$ is nonsingular. In that case EXACT_EIGENVECTOR returns the zero vector since that is the only solution of $(A - \mu \cdot I)\, x = 0$. Exact irrational eigenvalues may also produce the zero vector if ***DERIVE*** is unable to determine that certain complicated irrational expressions are equivalent to 0. Moreover, nonnumeric or irrational eigenvalues are likely to exhaust memory or yield extremely bulky eigenvector expressions. For these reasons, EXACT_EIGENVECTOR is effectively limited to a maximum of 4 x 4 matrices. APPROX_EIGENVECTOR is usually a better choice for numerical matrices larger than 3 x 3.

Exercise: Use the function EIGENVALUES to determine the exact eigenvalues of the following matrix, then use EXACT_EIGENVECTOR to determine the corresponding eigenvectors.

$$\begin{bmatrix} 7 & -1 & -2 \\ -1 & 7 & -2 \\ -1 & -1 & 6 \end{bmatrix}$$

APPROX_EIGENVECTOR(A, μ) approximates to an eigenvector of numerical matrix A corresponding to the eigenvalue that μ approximates. μ must be a number that is *not* an exact eigenvalue. The result is normalized to unit magnitude, making it unique to within a factor of plus or minus one for eigenvalues of multiplicity one. Multiply the result by a parameter such as @1 if you want a more general eigenvector. To assess the accuracy of a result vector x, compare it to $A \cdot x \,/\, \mu$ to see how closely they agree. APPROX_EIGENVECTOR uses inverse iteration to approximate eigenvectors.

Exercise: Use CHARPOLY to find the characteristic polynomial of the symmetric matrix

$$\begin{bmatrix} -2 & 1 & 1 & 1 & 1 \\ 1 & -3 & -1 & 0 & -1 \\ 1 & -1 & 1 & 0 & -3 \\ 1 & 0 & 0 & 3 & 0 \\ 1 & -1 & -3 & 0 & 2 \end{bmatrix}$$

then plot the result to isolate its zeros. Refine these zeros using the **soLve** command in approximate mode, then use APPROX_EIGENVECTOR to approximate each of the corresponding eigenvectors.

The file VECTOR.MTH makes the following assignments for polar-cylindrical and spherical coordinate geometry matrices for use with the pre-defined vector calculus functions (see Sections 8.9 and 8.10):

$$\texttt{cylindrical} := \begin{bmatrix} r & \theta & z \\ 1 & r & 1 \end{bmatrix}$$

and

$$\texttt{spherical} := \begin{bmatrix} r & \theta & \Phi \\ 1 & r \cdot \texttt{SIN}(\Phi) & r \end{bmatrix}$$

The file also defines functions for finding other coordinate geometry matrices in orthogonal curvilinear coordinates. First, compute the **Jacobian matrix** of the transformation from curvilinear to Cartesian coordinates. Next, compute the covariant metric tensor from the Jacobian matrix. If this matrix is diagonal, the coordinates are orthogonal, and you can proceed to compute the geometry matrix from the curvilinear coordinates and this diagonal:

Assume the transformation from curvilinear to some Cartesian coordinates is given by

$$x = u(\theta_1, \ \theta_2, \ \ldots, \ \theta_m)$$

where x is a vector of n Cartesian coordinates, and u is a vector of n expressions depending on curvilinear coordinates θ_1 through θ_m, with $m \leq n$. ($m < n$ corresponds to a hyper-surface imbedded in a higher-dimensional space.)

JACOBIAN(u, Θ) simplifies to the Jacobian matrix of the above transformation, where $\Theta = [\Theta_1, \Theta_2, \ldots, \Theta_m]$. The Jacobian is the n x m matrix of partial derivatives of u, with $du_i/d\Theta_j$ located at row i and column j. For example, the transformation from parabolic to Cartesian coordinates is

$$x = \frac{w^2 - v^2}{2}$$

$$y = w \cdot v$$

Thus

```
JACOBIAN([(w^2 - v^2)/2, w v], [w, v])
```

displays as

$$\text{JACOBIAN}\left[\left[\frac{w^2 - v^2}{2},\ w \cdot v\right],\ [w,\ v]\right]$$

and simplifies to

$$\begin{bmatrix} w & -v \\ v & w \end{bmatrix}$$

COVARIANT_METRIC_TENSOR(A) simplifies to the **covariant metric tensor** of the Jacobian matrix A. For example,

$$\text{COVARIANT\_METRIC\_TENSOR}\begin{bmatrix} w & -v \\ v & w \end{bmatrix}$$

simplifies to

$$\begin{bmatrix} w^2 + v^2 & 0 \\ 0 & w^2 + v^2 \end{bmatrix}$$

A coordinate system is orthogonal only if the elements off the main diagonal of the covariant metric tensor of its Jacobian matrix are 0. Thus the parabolic coordinate transformation is indeed orthogonal.

GEOMETRY_MATRIX(Θ, *G*) simplifies to the geometry matrix for curvilinear coordinate vector Θ having a covariant metric tensor *G*. For example,

```
GEOMETRY_MATRIX([u, v], [ u^2 + v^2      0      ])
                        [     0      u^2 + v^2 ]
```

simplifies to

$$\begin{bmatrix} u & v \\ \sqrt{u^2 + v^2} & \sqrt{u^2 + v^2} \end{bmatrix}$$

The second row of the geometry matrix is just the square roots of the diagonal elements of the covariant tensor. In some cases, simplification of the square roots may lead to unwanted absolute values unless you use the **Declare Variable Domain** command to appropriately restrict the domain of some coordinate variables (see Section 4.10). For example with spherical coordinates, if *r* is declared nonnegative and Φ is declared between 0 and π,

```
GEOMETRY_MATRIX([r, θ, Φ], COVARIANT_METRIC_TENSOR(JACOBIAN
    ([r SIN(Φ) COS(θ), r SIN(Φ) SIN(θ), r COS(Φ)], [r, θ, Φ])))
```

simplifies to

$$\begin{bmatrix} r & \theta & \Phi \\ 1 & r \cdot \mathrm{SIN}(\Phi) & r \end{bmatrix}$$

instead of to

$$\begin{bmatrix} r & \theta & \Phi \\ 1 & |r \cdot \mathrm{SIN}(\Phi)| & |r| \end{bmatrix}$$

9.3 NUMERIC.MTH — Numerical Differentiation and Integration

The file NUMERIC.MTH implements approximate numerical first and second derivatives of expressions along with smoothing, numerical differentiation, and integration of data arrays. It is usually more appropriate to use the **approX** command rather than the **Simplify** command with these functions.

DIF_NUM(y, x, x_0, h) approximates to the partial derivative of expression y with respect to variable x at value x_0, using a centered finite difference with step size h.

Exercise: The error in approximating the derivative of SIN(x) the centered finite difference at $x=\hat{e}$ is

$$\lim_{x \to \hat{e}} \frac{d}{dx} \text{SIN(x)} - \text{DIF\_NUM(SIN(x), x, } \hat{e}\text{, h)}$$

Simplify this expression, and plot the result. Zoom way in, both horizontally and vertically, on the y intercept to determine the optimum h, below which roundoff errors become significant.

DIF2_NUM(y, x, x_0, h) approximates to the second partial derivative of expression y with respect to variable x at value x_0, using a centered finite difference with step size h.

Exercise: Adapt the previous exercise to the second derivative.

SMOOTH_VECTOR(v) simplifies to a smoothed copy of vector v in which each internal component is the mean of the corresponding component and the two adjacent components in v. The end components of the result are the same as in v.

SMOOTH_COLUMN(A, j) simplifies to a copy of matrix A in which column j is smoothed as described for the preceding SMOOTH_VECTOR function. To smooth all of the columns, simplify the expression SMOOTH_VECTOR(A`)`.

DIF_DATA(A) approximates to the first derivative of a numerical data matrix A. A must have two columns and at least three rows, with each successive row being a successive [x, y] coordinate pair. The result is an analogous data matrix wherein the second column is the approximate first derivative of y with respect to x at successive values of x. The x values can be unequally spaced. They can even be nonmonotonic, such as samples from a parametrically-defined curve. The formula is based on the derivatives of parabolas that fit three successive points. Such numerical differentiation tends to amplify roundoff or experimental errors, so it is often wise to apply the SMOOTH_COLUMN or SMOOTH_VECTOR function

one or more times before applying DIF_DATA, as described in the preceding paragraph.

Exercise: Plot the following experimental data matrix in a 2D-plot window:

$$\begin{bmatrix} -5 & -4.78 \\ -2.5 & -4.561 \\ -0.1 & -4.205 \\ 2.9 & -3.29 \\ 3.2 & -3.03 \\ 5.1 & -1.78 \\ 6.3 & 3.43 \\ 6.8 & 4.72 \end{bmatrix}$$

Next, approximate DIF_DATA of the matrix and then superimpose a plot of the result. If you can plot using multiple colors, use the **Options Color Auto** and **Options Color Plot** commands to plot the first matrix all in one color and the second matrix all in a second color (see Section 5.1.2). If you can only plot in a single color, split the plot window horizontally and plot one matrix above the other (see Section 2.13). Repeat this experiment after using SMOOTH_COLUMN to smooth the second column.

DIF2_DATA(*A*) is analogous to DIF_DATA(*A*), but it approximates the second derivative. The formula tends to be less accurate at the end points than at the internal points. The advice about possibly smoothing the data is even more relevant for DIF2_DATA.

Exercise: Continuing the previous exercise, approximate DIF2_DATA of the original matrix and then plot this result too — both with and without smoothing.

INT_DATA(*A*) is analogous to DIF_DATA(*A*), but uses the trapezoidal rule to approximate an antiderivative having the value 0 at the first data point. Such integration tends to smooth away roundoff and experimental errors, so it is usually not necessary to smooth data before applying INT_DATA.

Exercise: Apply INT_DATA to the above data matrix and then apply DIF_DATA to the result. Finally, compare plots of the original and final matrices.

9.4 DIF_APPS.MTH — Applications of Differentiation

The file DIF_APPS.MTH defines functions that automate some of the most common applications of differentiation. These applications include parametric, polar and implicit derivatives, curvature, centers-of-curvature, tangent lines, perpendicular lines, osculating circles, tangent planes, and surface normals. There are separate groups of functions for curves defined in Cartesian coordinates either explicitly, parametrically, or implicitly, and for curves explicitly defined in polar coordinates.

The following functions are for curves explicitly defined in Cartesian coordinates by an abscissa and an ordinate, where the ordinate is the value of the expression y that depends on the abscissa variable x:

CURVATURE(y, x) simplifies to the curvature of expression y in terms of variable x. If you want the curvature at a particular value of x, then substitute that value in the result or take the limit of the result as x approaches that value.

Exercise: Plot the expression 2 COS(x) and its curvature in a 2D-plot window. Then substitute 0 for x in the curvature and simplify the result to determine the curvature at $x=0$.

CENTER_OF_CURVATURE(y, x) simplifies to the Cartesian coordinate vector of the center of curvature of expression y in terms of its variable x.

Exercise: Plot the parabola $x^2/4 - 2$ and its center of curvature in a 2D-plot window.

TANGENT(y, x, x_0) simplifies to the line tangent to expression $y(x)$ at $x=x_0$. The result will be linear in variable x. TANGENT(y, x, ∞) simplifies to the asymptote of $y(x)$ as x approaches ∞.

Exercise: Plot the parabola $x^2/4 - 2$ and its tangent line at $x=1$ in a 2D-plot window.

PERPENDICULAR(y, x, x_0) simplifies to the line perpendicular to expression y at $x=x_0$. The result will be linear in variable x.

Exercise: Plot the parabola $x^2/4 - 2$ in a 2D-plot window. Then simplify the expression PERPENDICULAR($x^2/4 - 2$) to determine an expression for the perpendicular to this parabola at $x=x_0$ where x_0 is a variable. Next, letting *n* denote the specific expression number of this result, simplify VECTOR(#*n*, x_0, −3, 3, 6/7) to determine a family of lines perpendicular to the parabola at eight equally-spaced values of x_0. Finally, superimpose a plot of the resulting vector of

expressions on the plotted parabola. If the lines do not look perpendicular to the parabola, use the **aXes** command to adjust the aspect ratio (see Section 5.1.2).

OSCULATING_CIRCLE(y, x, Θ) simplifies to a parametric representation of the circle that osculates expression y at its variable x, using Θ as the angular parameter.

Exercise: Plot 2 COS(x) in a 2D-plot window and then simplify OSCULATING_CIRCLE(2 COS(x)) to determine how the circle depends on x. Next, substitute 0 for x and simplify to determine the parametric circle that osculates at $x=0$. Finally, superimpose a plot of the resulting circle. If the circle does not look round, use the **aXes** command to adjust the aspect ratio (see Section 5.1.2).

The following functions are for curves parametrically defined in Cartesian coordinates by a vector of the form $v = [x, y]$, where x and y are expressions that depend on the parameter t:

PARA_DIF(v, t, n) simplifies to $d^n y/dx^n$ in terms of parametric variable t. n must be a nonnegative integer, and it defaults to 1. To learn how to program such defaults, inspect PARA_DIF's definition in the file DIF_APPS.

Exercise: Parametrically plot the unit cycloid

```
[t - SIN(t), 1 - COS(t)]
```

then substitute $t=1$ and superimpose a plot of this point. Next, use PARA_DIF to find the cycloid's second derivative, d^2y/dx^2, in terms of t. Finally, substitute 1 for t in the result and simplify to determine the second derivative at the plotted point.

PARA_CURVATURE(v, t) similarly simplifies to the curvature of parametric curve v in terms of parametric variable t.

Exercise: Use PARA_CURVATURE to find how the curvature of the unit cycloid of the previous exercise depends on t. Then substitute 1 for t and simplify.

PARA_CENTER_OF_CURVATURE(v, t) similarly simplifies to a Cartesian coordinate vector of the center of curvature of v in terms of parametric variable t.

Exercise: Use PARA_CENTER_OF_CURVATURE to find how the center of curvature of the unit cycloid of the previous exercise depends on t. Then superimpose a plot of this curve on a plot of the cycloid.

PARA_TANGENT(v, t, t_0, x) simplifies to an expression linear in variable x. This result is the y coordinate of the line tangent to parametric curve v at parameter value $t=t_0$.

Exercise: Use PARA_TANGENT to find the line tangent to the unit cycloid of the previous exercise at $t=1$. Then superimpose a plot of this line on a plot of the cycloid.

PARA_PERPENDICULAR(v, t, t_0, x) similarly simplifies to the line perpendicular to parametric curve v at parameter value $t=t_0$.

Exercise: Use PARA_PERPENDICULAR to find the line perpendicular to the unit cycloid at $t=1$. Then superimpose a plot of this line on a plot of the cycloid.

PARA_OSCULATING_CIRCLE(v, t, t_0, Φ) simplifies to a parametric representation of the circle that osculates parametric curve v at parameter value $t=t_0$. Variable Φ is the angular parameter of this circle.

Exercise: Use PARA_OSCULATING_CIRCLE to find the circle osculating the unit cycloid at $t=1$. Then superimpose a plot of this circle on a plot of the cycloid.

The following functions are for curves explicitly defined in polar coordinates by an expression r that depends on the angle Θ:

POLAR_DIF(r, Θ, n) simplifies to $d^n y/dx^n$ in terms of variable Θ. n must be a nonnegative integer, and it defaults to 1.

POLAR_CURVATURE(r, Θ) simplifies to the curvature of expression r in terms of variable Θ.

POLAR_CENTER_OF_CURVATURE(r, Θ) simplifies to the Cartesian coordinates of the center of curvature of expression r, in terms of variable Θ.

POLAR_TANGENT(r, Θ, Θ_0, x) simplifies to an expression linear in variable x. This result is the y coordinate of the line tangent to expression r at angle $\Theta=\Theta_0$.

POLAR_PERPENDICULAR(r, Θ, Θ_0, x) similarly simplifies to the Cartesian line perpendicular to expression r at angle $\Theta=\Theta_0$.

POLAR_OSCULATING_CIRCLE(r, Θ, Θ_0, Φ) simplifies to the Cartesian parametric circle that osculates polar curve r at angle $\Theta=\Theta_0$. Variable Φ is used as the angular circle parameter.

Exercise: For $r=\Theta$, determine expressions for the polar first derivative, curvature, and center of curvature. Superimpose plots of these in polar coordinates. Switching to Cartesian coordinates, also superimpose plots of the tangent, perpendicular and osculating circle at $\Theta_0=1$.

The following functions are for curves implicitly defined in Cartesian coordinates by an equation of the form $u=0$, where u is an expression that depends on the variables x and y. The values of the arguments for x and y must be unassigned variables. Do not declare y to be a function. The results are unique only modulo $u(x,y)=0$ or $u(x_0,y_0)=0$, so the symbolic results may differ from results derived by other methods. However, the numerical results from substituting a pair of numbers that satisfy $u(x,y)=0$ or $u(x_0,y_0)=0$ should be the same, within roundoff error:

IMP_DIF(u, x, y, n) simplifies to the implicit derivative d^ny/dx^n. n must be a positive integer, and it defaults to 1. In most cases the result depends on both x and y rather than x or y alone. This is acceptable for some applications. If not, you can sometimes use the original equation $u=0$ to eliminate either y or x from the derivative. Moreover, you can always substitute a particular value of x or of y into u, then solve for the other variable. This may require approximate mode. Either way, the resulting numerical values of x and y can then be substituted into the derived general implicit derivative.

Exercise: Given y = COS(x y) + x, simplify IMP_DIF(y – COS(x y) – x) to determine a formula for dy/dx in terms of x and y. Then to determine its value on the curve at $x=1$, substitute this value for x in the given equation and use the **soLve** command in approximate mode to determine the corresponding value of y. Finally, substitute this value for y and 1 for x in the simplified derivative, and simplify.

IMP_CURVATURE(u, x, y) simplifies to the curvature of the curve implicitly defined by $u=0$.

IMP_CENTER_OF_CURVATURE(u, x, y) simplifies to the center of curvature of the curve implicitly defined by $u=0$.

IMP_TANGENT(u, x, y, x_0, y_0) simplifies to an expression linear in variable x. The result is the y coordinate of the line tangent at $x=x_0$ and $y=y_0$ to the curve implicitly defined by $u=0$.

IMP_PERPENDICULAR(u, x, y, x_0, y_0) similarly simplifies to the line perpendicular at $x=x_0$ and $y=y_0$ to the curve implicitly defined by $u=0$.

IMP_OSCULATING_CIRCLE(u, x, y, x_0, y_0, Θ) simplifies to the circle that osculates at $x=x_0$ and $y=y_0$ the curve implicitly defined by $u=0$. The circle is defined parametrically in terms of variable Θ.

Exercise: Continuing the previous exercise, determine the curvature, center of curvature, tangent, perpendicular, and osculating circle at $x=1$.

The following functions implement applications of derivatives to surfaces defined implicitly by an equation of the form $u=0$, where u is an expression that depends on the three Cartesian coordinate variables comprising the vector v. Typically v is the vector $[x, y, z]$:

TANGENT_PLANE(u, v, v_0) simplifies to an expression that is linear in the three coordinate variables of v. This result defines the plane tangent to the surface $u=0$ at $v=v_0$: When equated to zero, the result is a linear relationship between the variables in v, and you can **soLve** this equation for any of its variables in terms of the others. Given any two components of v_0, you can substitute them into u, and then solve for the remaining component. Solving it may require switching to approximate mode.

Exercise: Determine expressions for the planes tangent to the surface

$$x + 2\cdot y + 3\cdot z = \hat{e}^{-(2\cdot x + y + z)}$$

at all points where $x=1$ and $y=-1$.

NORMAL_LINE(u, v, v_0, t) simplifies to a Cartesian coordinate vector that is linear in variable t. This result parametrically defines the line perpendicular to the surface $u=0$ at $v=v_0$. v is a vector of coordinate variables, and v_0 is a corresponding vector of coordinate values on the surface.

Exercise: Determine the normal lines through the same points on the same surface as the previous exercise.

9.5 INT_APPS.MTH — Applications of Integration

The file INT_APPS.MTH defines functions that automate various applications of integration. These include Laplace transforms and Fourier series together with arc lengths, areas, volumes, centroids, and inertia tensors, in Cartesian, cylindrical, and spherical coordinates.

LAPLACE(y, t, s) simplifies to the **Laplace transform** of expression y depending on variable t, using variable s as the independent variable in the transform domain. For example, LAPLACE(t^2, t, s) simplifies to $2/s^3$, if s has been declared positive. Finding the transform depends on symbolically integrating an improper integral. Convergence of this integral generally requires that s be declared positive. However if y has a factor of the form $\hat{e}^{kt}$, where k is a constant, then s must be declared greater than k in order for the integral to converge.

There are known Laplace transforms that LAPLACE can not determine, so look in a table if LAPLACE does not return a closed form. If your reason for using Laplace transforms is to solve a first or second order ordinary differential equation, the files ODE1.MTH and ODE2.MTH have functions that more directly automate the process.

Exercise: After declaring the variables a and s positive, use LAPLACE to determine the Laplace transform of $\hat{e}^{-at}$ SIN(t).

To determine the inverse Laplace transform of an expression u depending on the transform variable s, use the **Expand** command to determine a partial fraction expansion of u with respect to s (see Section 4.7), and then look up the inverse transform of each term in a table.

FOURIER(y, t, t_1, t_2, n) simplifies to a truncated **Fourier series** of expression y depending on variable t varying from t_1 to t_2. n is a nonnegative integer, and the series is truncated to the nth harmonic, having the form

$$b_0 + \sum_{k=1}^{n} \left[a_k \, \text{SIN}\left[\frac{2 \cdot \pi \cdot k \cdot t}{t_2 - t_1}\right] + b_k \, \text{COS}\left[\frac{2 \cdot \pi \cdot k \cdot t}{t_2 - t_1}\right]\right]$$

where b_0 through b_n and a_1 through a_n depend on the particular expression y. Try using the **approX** command if the **Simplify** command does not yield a closed form.

Exercise: Simplify the expression

```
VECTOR(FOURIER(t, t, -2, 2, n), n, 4)
```

and then plot the resulting matrix from about t=–4 through 4 to study the convergence to a sawtooth function.

Most of the remaining functions in this section involve an optional last parameter μ describing how a density varies throughout the region. Examples of this density include mass, charge, and momentum. The default value is $\mu=1$.

ARC_LENGTH(y, x, x_1, x_2, μ) simplifies to the integral of $\mu(x)$ along the arc described by expression y depending on variable x varying from x_1 to x_2. Omit μ to determine arc length. Let μ = [x, y] to determine the first moment. (The first moment divided by the arc length is the centroid.)

Exercise: Use ARC_LENGTH to determine the arc length of x^2 from $x=a$ to b.

POLAR_ARC_LENGTH(r, Θ, Θ_1, Θ_2, μ) simplifies to the integral of $\mu(\Theta)$ along the arc described by polar expression r depending on angular variable Θ varying from Θ_1 to Θ_2. Omit μ to determine the arc length.

Exercise: Polar plot the polar cardiod 1 – COS(Θ) from $\Theta=-\pi$ to π, and use POLAR_ARC_LENGTH to determine its arc length.

PARA_ARC_LENGTH(v, t, t_1, t_2, μ) simplifies to the integral of $\mu(t)$ over the curve parametrically defined by a Cartesian vector v depending on a variable t that varies from t_1 to t_2. v may have any number of components, such as two for a plane curve or three for a space curve. Omit μ to determine the arc length.

Exercise: Determine the arc length of the helix [r COS(Θ), r SIN(Θ), h Θ] from $\Theta=-\pi$ to π.

The remaining definitions in this section involve multiple integrals. You may have to finish up with the **approX** command if ***DERIVE*** cannot determine an exact closed form for an integral. Also changing the order of integration or the coordinate system may enable a closed-form solution to be found. This is important because successive symbolic integrations tend to reduce singularities and smooth the integrand, making any subsequent numerical integrations faster and more accurate. Moreover, the computing time tends to increase by more than an order of magnitude for each nested integral that must be done numerically.

AREA(x, x_1, x_2, y, y_1, y_2, μ) simplifies to the integral of $\mu(x, y)$ over the region from variable $x=x_1$ to x_2 and variable $y=y_1(x)$ to $y_2(x)$. You may have to partition a region into several regions and/or permute the roles of x and y to describe the region this way. Omit the last argument if you merely want the area. If y_1 ever exceeds y_2 in the interval $x_1 \le x \le x_2$, then subdivide the integral at such places and interchange y_1 and y_2 so that $y_1 \le y \le y_2$ throughout each interval. However, in some such cases it may suffice to just let μ = SIGN($y_2 - y_1$). Consequently, you can try that first to see if it yields a closed-form result.

AREA_CENTROID(x, x_1, x_2, y, y_1, y_2, μ) similarly computes the centroid of density $\mu(x,y)$. Omit the last argument if you merely want the areal centroid.

AREA_INERTIA(x, x_1, x_2, y, y_1, y_2, μ) similarly computes the inertia tensor of density $\mu(x,y)$, represented as a symmetric matrix of the form:

$$\begin{bmatrix} I_{xx} & -I_{xy} \\ -I_{xy} & I_{yy} \end{bmatrix}$$

The off-diagonal elements are minus the product of inertia. Omit the last argument if you merely want the areal inertia tensor. The **polar moment** of inertia is $I_{xx}+I_{yy}$. The **radius of gyration** is the polar moment of inertia divided by the area.

Exercise: Determine the area, centroid, inertia tensor, and polar inertia of the region $0 \le x \le b$ and $\text{SIN}(x) \le y \le 1+x^2$.

POLAR_AREA(r, r_1, r_2, Θ, Θ_1, Θ_2, μ) simplifies to the integral of $\mu(\Theta)$ over the region from variable $\Theta=\Theta_1$ to Θ_2 and variable $r=r_1(\Theta)$ to $r_2(\Theta)$. Omit the last argument if you merely want the area. If $r_1 > r_2$ anywhere in $\Theta_1 \le \Theta \le \Theta_2$, read the discussion of function AREA above.

Exercise: Plot using polar coordinates the cardiod $1 - \text{COS}(\Theta)$ from $\Theta=-\pi$ to π, then use POLAR_AREA to determine its area.

SURFACE_AREA(z, x, x_1, x_2, y, y_1, y_2, μ) simplifies to the integral of $\mu(x, y)$ over the surface defined by $z(x, y)$ for variable $x=x_1$ to x_2 and variable $y=y_1(x)$ to $y_2(x)$. A region may have to be partitioned into several subregions and/or the roles of x, y, and z permuted in order to describe a region in this way. Omit the last argument if you merely want the surface area. Let $\mu = [x, y, z]$ to compute the centroid, and similarly adapt the definition of VOLUME_INERTIA below to determine the inertia tensor. If $y_1 > y_2$ anywhere in $x_1 \le x \le x_2$, read the discussion of function AREA above.

Exercise: For the region $0 \le x \le 1$ and $x-1 \le y \le 1-x$, determine the area of the surface $z = 1 - x^2 + y$.

VOLUME(x, x_1, x_2, y, y_1, y_2, z, z_1, z_2, μ) simplifies to the integral of $\mu(x, y, z)$ over the region from variable $x=x_1$ to x_2, variable $y=y_1(x)$ to $y_2(x)$, and variable $z=z_1(x, y)$ to $z_2(x, y)$. You may have to partition a region into several regions and/or permute the roles of x, y, and z to describe a region this way. Omit the last argument if you merely want the volume. If $y_1 > y_2$ or $z_1 > z_2$ anywhere in $x_1 \le x \le x_2$, read the discussion of function AREA above.

VOLUME_CENTROID(x, x_1, x_2, y, y_1, y_2, z, z_1, z_2, μ) similarly computes the centroid of density $\mu(x, y, z)$. Omit the last parameter if you merely want the volumetric centroid.

VOLUME_INERTIA(x, x_1, x_2, y, y_1, y_2, z, z_1, z_2, μ) similarly computes the inertia tensor of density $\mu(x, y, z)$, represented as a symmetric matrix of the form.

$$\begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}$$

The off-diagonal elements are minus the products of inertia. Omit the last parameter if you merely want the volumetric inertia.

Exercise: Determine the mass, center of mass, and mass inertia tensor of the region $0 \le x \le 1$, $x \le y \le \hat{e}^x$, and $0 \le z \le x+y$ having mass density $\mu = x+y+z$.

SPHERICAL_VOLUME(r, r_1, r_2, Θ, Θ_1, Θ_2, Φ, Φ_1, Φ_2, μ) simplifies to the integral of $\mu(r, \Theta, \Phi)$ over the region from variable $r=r_1$ to r_2, variable $\Theta=\Theta_1(r)$ to $\Theta_2(r)$, and variable $\Phi=\Phi_1(r,\Theta)$ to $\Phi_2(r,\Theta)$. Θ is the colongitude measured in radians counterclockwise in the x-y plane from the positive x-axis. Φ is the colatitude measured in radians from the positive z-axis. Beware that some texts define spherical coordinates differently, such as interchanging the definitions of Θ and Φ.

You must modify the order of integration in the definition if r_1 or r_2 depend upon the angles or if Θ_1 or Θ_2 depends on Φ. Omit the last argument if you merely want the volume of this region. If $\Theta_1 > \Theta_2$ or $\Phi_1 > \Phi_2$ anywhere in $r_1 \le r \le r_2$, read the discussion of function AREA above.

Let μ equal

```
[r·SIN(Φ)·COS(θ), r·SIN(Φ)·SIN(θ), r·COS(Φ)]
```

to determine the centroid, or similarly adapt the definition of VOLUME_INERTIA to determine the inertia tensor.

Exercise: Determine the integral of $r^2\Theta+\Phi$ over the volume of the region bounded by $r_1=0$, $r_2=1$, $\Theta_1=0$, $\Theta_2=r$, $\Phi_1=0$, and $\Phi_2=r+\Theta$.

> **Hint**: Use $r^2\Theta+\Phi$ for μ.

CYLINDRICAL_VOLUME(r, r_1, r_2, Θ, Θ_1, Θ_2, z, z_1, z_2, μ) simplifies to the integral of $\mu(r, \Theta, z)$ over the region from variable $z=z_1$ to z_2, variable $\Theta=\Theta_1(z)$ to $\Theta_2(z)$, and variable $r=r_1(\Theta, z)$ to $r_2(\Theta, z)$. You must modify the order of integration in the definition if Θ_1 or Θ_2 depends on r or if z depends on r or Θ. Omit the last argument if you merely want the volume of this region. If $\Theta_1 > \Theta_2$ or $r_1 > r_2$ anywhere in $z_1 \le z \le z_2$, read the discussion of function AREA above. Let μ equal

```
[r·COS(θ), r·SIN(θ), z]
```

to determine the centroid, and similarly adapt the definition of VOLUME_INERTIA to determine the inertia tensor.

VOLUME_OF_REVOLUTION(y, x, x_1, x_2) simplifies to the volume of an expression $y(x)$ from variable $x=x_1$ to x_2, revolved about the x-axis.

Exercise: The **Horn of Gabriel** is formed by revolving $1/x$ around the x-axis from $x=1$ to ∞. Determine its volume.

AREA_OF_REVOLUTION(y, x, x_1, x_2) simplifies to the surface area of an expression $y(x)$ from variable $x=x_1$ to x_2, revolved about the x-axis.

Exercise: If each unit of paint volume covers 1000 units of surface, determine how much paint it takes to paint the inside of the Horn of Gabriel described in the previous exercise. Resolve the apparent paradox revealed by these two exercises.

VOLUMEY_OF_REVOLUTION(y, x, x_1, x_2) simplifies to the volume of an expression $y(x)$ from variable $x=x_1$ to x_2, revolved about the y-axis. If your volume of revolution has a hole of revolution given by expression y_2, then compute

```
VOLUMEY_OF_REVOLUTION(y, x, x1, x2) -
                      VOLUMEY_OF_REVOLUTION(y2, x, x1, x2)
```

Exercise: Determine the volume of rotating $1/x$ about the y-axis from $x=0$ to 1, and resolve the apparent difference from the volume of the above Horn of Gabriel.

AREAY_OF_REVOLUTION(y, x, x_1, x_2) simplifies to the area of revolution of an expression $y(x)$ from variable $x=x_1$ to x_2, revolved about the y-axis.

Exercise: Approximate the surface area from revolving $y = x^5 + x + 1$ about the y-axis from $x=0$ to 1. Could this problem be done by interchanging x and y, solving for $y(x)$, and then using the function AREA_OF_REVOLUTION?

9.6 ODE1.MTH — First Order Ordinary Differential Equations

The file ODE1.MTH defines functions for finding exact solutions to first order ordinary differential equations. The file begins with methods taught in second-year calculus and elementary differential equations courses. Following the elementary methods are advanced methods that may succeed when the more elementary methods are inapplicable.

Functions with names having the suffix GEN return a *general* solution in terms of a symbolic constant. Functions with names without the suffix GEN return a *specific* solution when given numeric initial conditions, or they return a general solution in terms of symbolic initial conditions when given unassigned variables for the initial conditions.

With either type of function, the result is usually an algebraic equation implicitly defining the dependent variable in terms of the independent variable. This equation may contain integrals that ***DERIVE*** could not determine exactly. However, such implicit solutions are still considered solutions of differential equations even if integrals remain. If an implicit solution does not contain an integral involving the dependent variable, you can use the **soLve** command in an attempt to find one or more explicit solutions.

In this and the following section as well as in the corresponding utility files, the variable x denotes the independent variable, and the variable y denotes the dependent variable. Also the variable x_0 denotes the initial value of x, and the variable y_0 denotes the corresponding initial value of y. Do *not* use x or y as function names. In this and the following section, dy/dx is abbreviated as y' for descriptive purposes only. In ***DERIVE*** the apostrophe or single-quote mark can *not* be used to denote derivatives.

If you want to find a specific solution for symbolic or numeric initial conditions, it is best to use a function that directly returns specific solutions. Otherwise, after finding a general solution, you must substitute x_0 and y_0 into the general solution, solve for c, and then substitute that value of c back into the general solution.

Elementary Methods

DSOLVE1_GEN(p, q, x, y, c) simplifies to a general solution of an equation having the form

$$p(x, y) + q(x, y)\cdot y' = 0$$

in terms of symbolic constant c. Note that most first order differential equations can be put in the above form.

DSOLVE1_GEN can solve exact, linear, separable, homogeneous or generalized-homogeneous equations, and equations having an integrating factor that depends only on x or only on y. In other words, DSOLVE1_GEN is a general-purpose function that can solve most first order equations that occur in second-year calculus or elementary differential equations courses.

If the equation is not one of the above forms, DSOLVE1_GEN returns the word "inapplicable". In this case, consider seeing if one of the advanced methods described in the next subsection is applicable to the equation.

DSOLVE1(p, q, x, y, x_0, y_0) is similar to DSOLVE1_GEN, but simplifies to a specific solution for the initial condition $y=y_0$ at $x=x_0$. These initial conditions can be numbers, variables, or general expressions.

For example, to solve the differential equation $2 \cdot x \cdot y + (1 + x^2) \cdot y' = 0$ for the initial condition $y=1$ at $x=0$, enter and simplify the expression

```
DSOLVE1(2·x·y, 1 + x^2, x, y, 0, 1)
```

to produce the implicit algebraic solution

```
x^2·y + y - 1 = 0
```

To verify that it satisfies the initial condition, substitute $x=0$ and $y=1$ into this implicit solution, and then see that it simplifies to $0=0$.

The implicit differentiation function IMP_DIF in the utility file DIF_APPS.MTH (see Section 9.4) may be helpful for verifying an implicit solution of a differential equation. For example, to verify the above implicit solution, use the **Transfer Load Utility** command to load DIF_APPS.MTH, enter the expression

```
2·x·y + (1 + x^2)·IMP_DIF(x^2·y + y - 1, x, y) = 0
```

and then see that it simplifies to $0=0$.

If it is necessary to obtain an explicit solution of a differential equation, use the **soLve** command to solve the implicit solution for y. For example, solving the above implicit solution for y gives

$$y = \frac{1}{x^2 + 1}$$

To verify that this explicit solution satisfies the differential equation, substitute the solution for y in the *un*simplified difference of the two sides of the differential

equation and see that the result simplifies to 0. For example, substituting $1/(x^2+1)$ for y in

$$2 \cdot x \cdot y + (1 + x^2) \cdot \frac{d}{dx} y$$

and then simplifying the result gives 0.

The remaining functions in this subsection are for solving specific forms of differential equations. They are included primarily for educational purposes.

SEPARABLE_GEN(p, q, x, y, c) simplifies to a general implicit solution of an equation having the form $y' = p(x) \cdot q(y)$, where $p(x)$ is an expression free of y and $q(y)$ is an expression free of x. The right side may require factoring and/or applying various trigonometric, logarithmic, or exponential transformations to transform it to separable form.

Exercise: Find and verify a general solution of the equation $y' = \hat{e}^{x+y} + \hat{e}^{y}$.

> **Hint**: Trivially factor the right side of the equation to obtain the equivalent separable equation $y' = \hat{e}^{y}(\hat{e}^{x} + 1)$.

SEPARABLE(p, q, x, y, x_0, y_0) is similar to SEPARABLE_GEN, but simplifies to a specific solution for the initial condition $y=y_0$ at $x=x_0$.

LINEAR1(p, q, x, y, x_0, y_0) simplifies to an explicit solution of the linear monic differential equation $y' + p(x) \cdot y = q(x)$. A differential equation is **monic** if the coefficient of its highest order derivative is 1. Note that this equation does not have to be linear in x: only linear in y and its derivative.

Exercise: Find and verify a specific solution of the equation $x \cdot y' - 4 \cdot y + 2 \cdot x^2 + 4 = 0$ for the initial condition $y=1$ at $x=1$.

LINEAR1_GEN(p, q, x, y, c) is similar to LINEAR1, but simplifies to a general solution in terms of symbolic constant c.

HOMOGENEOUS(r, x, y, x_0, y_0) simplifies to an implicit solution of an equation having the form $y' = r(x, y)$ if r is homogeneous, otherwise it returns "inapplicable". In this context, an expression is **homogeneous** if substituting $k\,y$ for y in the expression yields an equivalent expression. Often homogeneous expressions consist of a ratio of two polynomials in x and y, with exponents of x and y totaling the same constant in every term. The arguments of any irrational

subexpressions must also be homogeneous. For example, $r = (x^2 \cdot y - x \cdot y^2)/(x^3 - y^3) + \text{SIN}(x/y)$ is homogeneous.

HOMOGENEOUS_GEN(r, x, y, c) is similar to HOMOGENEOUS, but simplifies to a general solution in terms of symbolic constant c if r is homogeneous.

Exercise: Use HOMOGENEOUS_GEN to find a general solution of the equation

$$y' = \frac{y \cdot \hat{e}^{y/x} - x}{x \cdot \hat{e}^{y/x}}$$

or to determine that it is inhomogeneous.

EXACT(p, q, x, y, x_0, y_0) simplifies to an implicit solution of an equation of the form $p(x, y) + q(x, y)\, y' = 0$ if this differential equation is **exact**, otherwise it returns "inapplicable". This equation is exact if and only if $dp/dy - dq/dx$ is equivalent to 0.

EXACT_GEN(p, q, x, y, c) is similar to EXACT, but simplifies to a general solution in terms of symbolic constant c.

Exercise: Use EXACT_GEN to find a general solution of the equation

$$\hat{e}^{-x} \cdot \text{SIN}(y) - (y + \hat{e}^{-x} \cdot \text{COS}(y)) \cdot y' = 0$$

or to determine that it is inexact.

INTEGRATING_FACTOR(p, q, x, y, x_0, y_0) simplifies to an implicit solution of an equation of the form $p(x, y) + q(x, y) \cdot y' = 0$ if this equation is exact when multiplied by an integrating factor free of x or free of y, otherwise it returns "inapplicable".

INTEGRATING_FACTOR_GEN(p, q, x, y, c) is similar to INTEGRATING_FACTOR, but simplifies to a general solution in terms of symbolic constant c.

Exercise: Find and verify a general solution of the equation

$$4 \cdot x \cdot y + 3 \cdot y^2 - x + (x^2 + 2 \cdot x \cdot y) \cdot y' = 0$$

Advanced Methods

MONOMIAL_TEST(p, q, x, y) simplifies to an expression. If this expression is a monomial of the form $x^m \cdot y^n$, it is an integrating factor of the equation $p(x, y) + q(x, y) \cdot y' = 0$. Thus if MONOMIAL_TEST returns a monomial, multiply the equation by this monomial and solve the resulting exact differential equation using EXACT or DSOLVE1.

Exercise: Use MONOMIAL_TEST to find an integrating factor of the equation

$$2 \cdot y^3 - 5 \cdot x \cdot y + (y^2 \cdot x - 3 \cdot x^2) \cdot y' = 0$$

and then use this integrating factor to find a general solution of the equation.

BERNOULLI(p, q, k, x, y, x_0, y_0) simplifies to an implicit solution of the **Bernoulli equation** $y' + p(x) \cdot y = q(x) \cdot y^k$, where k is a constant.

BERNOULLI_GEN(p, q, k, x, y, c) is similar to BERNOULLI, but simplifies to a general solution in terms of symbolic constant c.

Exercise: Find and verify a general solution of the equation

$$dz/dt = (\hat{e}^t - z^2)/z$$

GEN_HOM(r, x, y, x_0, y_0) simplifies to an implicit solution of an equation of the form $y' = r(x, y)$ if r is **generalized homogenous**, otherwise it returns "inapplicable". r is a generalized homogeneous expression if and only if

$$\frac{t \cdot d(t \cdot r)/dx}{d(t \cdot r)/dy}$$

is free of x and y, where $t = x/y$.

GEN_HOM_GEN(r, x, y, c) is similar to GEN_HOM, but simplifies to a general solution in terms of symbolic constant c.

Exercise: Find and verify a general solution of the equation

$$y^2 \cdot x^4 + 1 - x^3 \cdot y' = 0$$

FUN_LIN_CCF(r, p, q, k, x, y, x_0, y_0) simplifies to an implicit solution of an equation of the form $y' = r(u)$, where r is a function of linear constant coefficients: $u = p \cdot x + q \cdot y + k$, with p, q, and k independent of x and y.

FUN_LIN_CCF_GEN(r, p, q, k, x, y, c) is similar to FUN_LIN_CCF, but simplifies to a general solution in terms of symbolic constant c.

Exercise: Find and verify a general solution of the equation

$$y' = 1 + \mathrm{SEC}(1 + y + x)$$

LIN_FRAC(r, a, b, d, p, q, k, x, y, x_0, y_0) simplifies to an implicit solution of a **linear fractional** equation $y' = r(u)$, where $u = (a \cdot x + b \cdot y + d)/(p \cdot x + q \cdot y + k)$. However if $q \cdot a - p \cdot b = 0$, instead use FUN_LIN_CCF described above.

LIN_FRAC_GEN(r, a, b, d, p, q, k, x, y, c) is similar to LIN_FRAC, but simplifies to a general solution in terms of symbolic constant c.

Exercise: Find and verify a general solution of the equation

$$y' = (x + y - 1)/(x - y)$$

ALMOST_LIN(r, b, p, q, x, y, x_0, y_0) simplifies to an implicit solution of an equation of the form $r(x, y) \cdot y' + p(x) \cdot b(y) = q(x)$, if the equation is **almost-linear**, otherwise it returns "inapplicable". The differential equation is almost-linear if $(db/dy)/r(x, y)$ is free of y.

ALMOST_LIN_GEN (r, b, p, q, x, y, c) is similar to ALMOST_LIN, but simplifies to a general solution in terms of symbolic constant c.

Exercise: Find and verify a general solution of the equation

$$y^2 \cdot y'/x - y^3 = \hat{e}^x$$

All of the above exercise equations are **quasi-linear**, meaning linear in y'. If an equation is not quasi-linear, represent y' by a variable v and try solving for v to transform the equation into one or more alternate quasi-linear equations. Alternatively, the following method is directly applicable to some equations that are not quasi-linear, and this method may be preferable even if you can solve for v:

CLAIRAUT(p, q, x, y, v, c) solves the generalized **Clairaut equation**, which has the form $p(x \cdot v - y) = q(v)$, where p and q are any functions and v is a variable representing y'. CLAIRAUT returns a vector: The first component is a general solution containing an arbitrary constant c. The second component is an equation that you can attempt to solve algebraically for v. If you succeed, substitute this expression for v in the differential equation to obtain a **singular solution**.

For example, to solve the differential equation $(x \cdot y' - y)^2 = (y')^2 + 1$, enter

```
CLAIRAUT((x·v - y)^2, v^2 + 1)
```

which simplifies to

$$[c^2 \cdot x^2 - 2 \cdot c \cdot x \cdot y + y^2 - c^2 - 1 = 0,\quad 2 \cdot (v \cdot x^2 - x \cdot y - v) = 0]$$

Solving the second component for v in terms of x and y gives

$$v = \frac{x \cdot y}{x^2 - 1}$$

Substituting the right side of this result for y' in the original differential equation gives the implicit singular solution

$$\frac{y^2}{(x^2 - 1)^2} = \frac{x^2 \cdot y^2}{(x^2 - 1)^2} + 1$$

9.7 ODE2.MTH — Second Order Ordinary Differential Equations

The file ODE2.MTH defines functions for finding exact solutions of second order ordinary differential equations. Sometimes more than one method may be applicable. The most easily applied methods are described first, beginning with direct methods that produce an explicit solution and only require recognizing a pattern. In contrast, later methods generally give implicit solutions or require more steps, such as solving a sequence of two first order differential equations. The file ODE1.MTH (see Section 9.6) can be used to help solve these first order equations.

The notation used here and in the file ODE2.MTH is the same as that described near the beginning of the previous section. Also $y=y_2$ at $x=x_2$ is used as a second boundary condition, and $y'=v_0$ at $x=x_0$ is used as a second initial condition. d^2y/dx^2 is abbreviated as y'' for descriptive purposes only. In ***DERIVE*** the double-quote mark can *not* be used to denote second derivatives.

DSOLVE2(p, q, r, x, c_1, c_2) simplifies to an explicit general solution of the linear second order ordinary differential equation

$$y'' + p(x)\cdot y' + q(x)\cdot y = r(x)$$

in terms of arbitrary constants c_1 and c_2. Note that the last two arguments can be omitted if they are variables and you are satisfied with the names c_1 and c_2.

If no method applies or the equation cannot be converted to an equivalent one having a p and a q that are independent of x, DSOLVE2 returns the word "inapplicable".

DSOLVE2 can easily find a solution if p and q are numeric constants. When q is a symbolic constant, the result will often have a more appropriate form (sinusoidal versus exponential) if q has been declared positive or negative.

A solution may contain indefinite integrals involving $r(x)$ for which ***DERIVE*** can not determine a closed-form antiderivative. To verify such a result, substitute the solution for y into the *un*simplified form

$$\left[\frac{d}{dx}\right]^2 y + p(x)\cdot\frac{d}{dx} y + q(x)\cdot y - r(x)$$

and then see that it simplifies to 0.

Exercise: Find a general solution of the equation

$$y'' - 2/x\cdot y' + (9 + 2/x^2)\cdot y = x\cdot\hat{e}^x$$

by entering and simplifing the expression

```
DSOLVE2(-2/x, 9+2/x^2, x·#e^x)
```

Then verify the solution.

If you want to find a specific solution for symbolic or numeric initial or boundary conditions, it is best to use DSOLVE2_IV or DSOLVE2_BV that directly return specific solutions. Otherwise, after finding a general solution using DSOLVE2, you must substitute these constraints into general solution, solve for c_1 and c_2, and then substitute those values back into the general solution.

DSOLVE2_BV(p, q, r, x, x_0, y_0, x_2, y_2) is similar to DSOLVE2, but simplifies to a specific solution that satisfies the boundary conditions $y=y_0$ at $x=x_0$ and $y=y_2$ at $x=x_2$.

Exercise: Find and verify a specific solution of the equation used in the previous exercise for the boundary conditions $y=1$ at $x=1$ and $y=0$ at $x=2$.

DSOLVE2_IV(p, q, r, x, x_0, y_0, v_0) is similar to DSOLVE2_BV, but simplifies to a specific solution that satisfies the initial conditions $y=y_0$ and $y'=v_0$ at $x=x_0$.

Exercise: Find and verify a specific solution of the equation used in the previous exercise for the initial conditions $y=y'=0$ at $x=1$.

AUTONOMOUS_CONSERVATIVE(q, x, y, x_0, y_0, v_0) simplifies to an implicit algebraic solution of an equation of the form $y''=q(y)$ having initial conditions $y=y_0$ and $y'=v_0$ at $x=x_0$. Equations of this form are **autonomous** because the variable x does not occur and **conservative** because y' does not occur. If instead of the initial condition $y'=v_0$ you have a second boundary condition $y=y_2$ at $x=x_2$, substitute x_2 for x and y_2 for y in the solution and solve for v_0, which you can then eliminate in favor of x_2 and y_2.

Exercise: Find and verify a specific solution of the equation $w''=w^{-1/2}$ for the initial conditions $w=w'=0$ at $x=0$.

> **Hint**: Declare w to be nonnegative (see Section 4.10) to help ***DERIVE*** find closed-forms for all the antiderivatives in the solution.

LIOUVILLE(p, q, x, y, c_1, c_2) simplifies to an implicit algebraic solution of the **Liouville equation**

$$y'' + p(x)\cdot y' + q(y)\cdot(y')^2 = 0$$

This solution is a linear combination of two arbitrary constants c_1 and c_2. Consequently, you can substitute two sets of initial or boundary conditions, and then solve two simultaneous linear equations for c_1 and c_2.

Exercise: Find and verify a general solution of the equation

$$y'' + y'/x - y \cdot (y')^2 = 0$$

To solve an equation missing the dependent variable y: $f(y'', y', x) = 0$, use a new variable v for y' and use v' for y'' to obtain a first order equation for v. If you can solve that first order equation to obtain an algebraic equation relating v, x, x_0, and v_0, then noting that $v=y'$, you can attempt to solve this subsequent first order equation to obtain an algebraic equation relating y, x, x_0, y_0, and v_0. The file ODE1.MTH may help solve the two successive first order equations.

Exercise: Find and verify a general solution of the equation

$$y' \cdot y'' = \sqrt{(1 + (y')^2)}$$

AUTONOMOUS(r, v) simplifies an expression for dv/dy, given an autonomous equation $y'' = r(y, v)$ with v representing y'. Equations of this form are **autonomous** because the independent variable x doesn't occur. If you can solve this resulting first order equation to obtain an algebraic equation relating v, y, v_0, and y_0, then noting that $v=y'$, this result is another first order differential equation, which you can then attempt to solve to obtain an algebraic equation relating y, x, x_0, v_0, and y_0. The file ODE1.MTH may help solve the two successive first order equations. If r is also independent of v, instead use the function AUTONOMOUS_CONSERVATIVE to solve the equation in one step.

Exercise: Find and verify a general solution of the equation

$$y \cdot y'' + (y')^2 + 1 = 0$$

EXACT2(p, q, x, y, v, c) simplifies to an algebraic equation relating the solution of the equation relating y, v, x, and an arbitrary symbolic constant c, provided the equation

$$p(x, y, v) \cdot y'' + q(x, y, v) = 0$$

is exact. Note that here the variable v is used to represent y'. You can then attempt to solve the resulting first order differential equation. If the equation is inexact, EXACT2 returns "inapplicable".

Exercise: Use EXACT2 to reduce the equation

$$(x^2 + 2 \cdot y^2 \cdot y') \cdot y'' + y + 3 \cdot x \cdot y' + 2 \cdot y \cdot (y')^3 = 0$$

to first order. Note that the methods in the file ODE_APPR.MTH can be used to solve the resulting first order equation approximately.

9.8 ODE_APPR.MTH — Approximate Solution of ODEs

The previous two sections describe methods for finding exact solutions to ordinary differential equations. Sometimes none of these methods is applicable to your equation or the results of these methods cannot be converted to an explicit form. Perhaps your differential equation is unsolvable in terms of ***DERIVE***'s functions, or you need to solve a system of differential equations. Don't give up if this happens!

The file ODE_APPR.MTH defines functions for solving most ordinary differential equations and systems of such equations using truncated series approximations and approximate numerical methods.

Series Solutions to First Order Equations and Systems

TAYLOR_ODE1(r, x, y, x_0, y_0, n) simplifies to an nth order truncated Taylor series solution of the equation $y' = r(x, y)$ given the initial condition $y=y_0$ and $x=x_0$. For example,

```
TAYLOR_ODE1(ê^(x·y), x, y, 0, 1, 4)
```

simplifies to

$$\frac{5 \cdot x^4}{12} + \frac{x^3}{2} + \frac{x^2}{2} + x + 1$$

To verify that this is an approximate series solution to the differential equation, substitute this series solution for y in the unsimplified difference between the two sides of the equation. Then use TAYLOR (see Section 7.3) to verify that the third order Taylor series approximation of that difference is 0. The reason for using a third rather than fourth order Taylor series approximation is that differentiation reduces the order of the truncated series by 1.

TAYLOR_ODE1 simplifies to ? if r is not sufficiently differentiable for an nth order truncated series. Try a smaller value for n if this happens.

PICARD(r, p, x, y, x_0, y_0) expands to an improved approximate series solution of the equation $y' = r(x, y)$, given the approximate series solution $p(x)$. Expanding with respect to x often gives a more useful form than just simplifying the expression.

The Picard method involves integrating $r(x, p(x))$. If no integrals remain in the simplified result, you can try another iteration using the improved approximation for p, and so on. If you have no better first approximation, use the constant y_0.

For example, suppose $y' = y^2+\sqrt{x}$ and $y=1$ at $x=0$. Use of TAYLOR_ODE1 indicates that the right side of the equation is sufficiently differentiable only for a first order truncated Taylor-series solution $x+1$. Consequently, using this as a first approximation,

```
PICARD(y^2+SQRT(x), x + 1, x, y, 0, 1)
```

expands to

$$\frac{x^3}{3} + x^2 + \frac{2\cdot x^{3/2}}{3} + x + 1$$

You can then use this second approximation as the second argument of PICARD to obtain a third approximation, and so on.

Note that the highest order terms generated by the Picard method are often incorrect. For example, the coefficient of the above x^3 term should be 1. Distrust terms whose coefficients are not the same for two successive iterates. For this reason and for efficiency, it is wise to discard all but the next higher order term after each iteration. For example, you should discard the above x^3 and x^2 terms for the next iteration.

If any iterate yields a result containing an integral, you can try approximating $r(x, p(x))$ to make it integrable.

Exercise: Generate several more correct terms of the solution of the above equation, and plot successive correct partial sums for comparison.

PICARD can also be used to find series solutions of *systems* of first order differential equations. Given a monic quasi-linear system of first order equations of the form

$$y_1' = r_1(x, y_1, y_2, \ldots, y_m)$$

$$y_2' = r_2(x, y_1, y_2, \ldots, y_m)$$

$$y_m' = r_m(x, y_1, y_2, \ldots, y_m)$$

let $r=[r_1, r_2, \ldots, r_m]$, $y=[y_1, y_2, \ldots, y_m]$, and $y=y_0=[y_{1,0}, y_{2,0}, \ldots, y_{m,0}]$ at $x=x_0$. Then PICARD(r, p, x, y, x_0, y_0) expands to an improved series solution of the system, given the series solution $p=[p_1, p_2, \ldots, p_m]$.

Exercise: Use PICARD at least twice to determine an approximate solution of the following system of equations:

$$z' = w \cdot z + \sqrt{x}$$

$$w' = w^2 + z^2$$

with $w=z=1$ at $x=0$.

TAYLOR_ODES(r, x, y, x_0, y_0, n) simplifies to the nth order truncated Taylor series solution of the above system of differential equations, provided vector r is sufficiently differentiable.

Exercise: Use TAYLOR_ODES to determine a fourth order truncated Taylor series solution of the system

$$z' = w \cdot z + \mathrm{COS}(x) - \mathrm{SIN}(2 \cdot x)/2$$

$$w' = w^2 + z^2 - \mathrm{SIN}(x) - 1$$

expanded about the initial values $z=0$ and $w=1$ at $x=0$. If you are familiar with the Taylor series of some elementary functions, try guessing the exact solution from the series, using substitution into the differential equations to verify your guess.

Series Solutions to Second Order Equations

TAYLOR_ODE2 (r, x, y, v, x_0, y_0, v_0, n) simplifies to an nth order truncated Taylor series solution of the equation $y'' = r(x, y, v)$. Here v represents y', and v_0 is the value of v at $x = x_0$. For example, TAYLOR_ODE2 ($2 \cdot x \cdot v + x^2 \cdot y + 3 \cdot x$, x, y, v, 0, 0, 1, 5) simplifies to

$$\frac{3 \cdot x^5}{10} + \frac{5 \cdot x^3}{6} + x$$

To verify that this is an approximate series solution to the differential equation, substitute this series solution for y and substitute the derivative of this expression for v into the unsimplified difference between the two sides of the equation. Then use TAYLOR (see Section 7.3) to verify that the third order Taylor series approximation of that difference is 0. The reason for using a third rather than fifth order Taylor series approximation is that the second derivative of the truncated series reduces its order by 2.

TAYLOR_ODE2 simplifies to ? if r is not sufficiently differentiable for an nth order truncated series. Try a smaller value for n if this happens.

Approximate Numerical Solutions to First Order Equations

EULER(r, x, y, x_0, y_0, h, n) approximates to a vector of $n+1$ solution points of the equation $y' = r(x, y)$ with $y=y_0$ at $x=x_0$ beginning with $x=x_0$ using a step size of h. EULER uses **Euler's method** to produce a vector of $n+1$ coordinate pairs of the form

$$[[x_0, y_0],\ [x_0+h, y_1],\ [x_0+2\cdot h, y_2],\ \ldots,\ [x_0+n\cdot h, y_n]]$$

This vector plots as a set of points that approximates the solution curve of the equation. For example, to generate five points of an approximate solution of the equation $y' = 26/(3+(x+y)^2)$ through $x=2$ from $x=1$ where $y=-2$, approximate

$$\text{EULER}\left[\frac{26}{3 + (x + y)^2},\ x,\ y,\ 1,\ -2,\ \frac{1}{4},\ 4\right]$$

giving

$$\begin{bmatrix} 1 & -2 \\ 1.25 & -0.375 \\ 1.5 & 1.35114 \\ 1.75 & 1.93520 \\ 2 & 2.32722 \end{bmatrix}$$

Exercise: Highlight, plot, and appropriately frame the above matrix. Use the **Options State** command to select large dots and the **Options Color Auto** command to select **No**. Then keep halving the step size and doubling the number of steps until two successive plots agree everywhere they share x values.

Euler's method has educational value as the simplest numerical method for solving ordinary differential equations, but it typically has the largest error of any standard method. In contrast, the more accurate Runge-Kutta method should be used for serious numerical solutions.

RK uses the classic fourth order **Runge-Kutta method** to solve a system of one or more simultaneous first order equations

RK(r, v, v_0, h, n) approximates to the solution of the *system* of first order differential equations $y_i' = r_i(x, y)$. r is the vector $[r_1, r_2, \ldots, r_n]$, v is the vector $[x, y_1, y_2, \ldots, y_m]$, v_0 is the corresponding vector of initial values, h is the desired step size, and n is the desired number of steps.

RK returns a matrix of successive approximate solution points, each row being a vector of numbers representing a successive value of vector v. To plot y_1 versus x, delete all but the first two columns. To plot y_2 versus x, delete all but the first

and third columns, etc. Alternatively, to plot y_2 versus y_1 — a parametric **phase-plane plot** — delete all but the second and third columns, etc. The following function automates the selection of column pairs:

EXTRACT_2_COLUMNS(A, j, k) simplifies to a matrix composed of the jth then kth columns of matrix A.

Exercise: Determine a 16-step solution over $0 \le x \le 2\pi$ to the problem in the exercise for TAYLOR_ODES, by approximating

$$\text{RK}\left(\left[w \cdot z + \cos(x) - \frac{\sin(2 \cdot x)}{2},\ w^2 + z^2 - \sin(x) - 1\right],\ [x,z,w],\ [0,0,1],\ \frac{2 \cdot \pi}{16},\ 16\right)$$

Then enter and simplify

```
EXTRACT_2_COL(#..., 1, 2)
EXTRACT_2_COL(#..., 1, 3)
EXTRACT_2_COL(#..., 2, 3)
```

where #... is the label number of the expression you just approximated. Finally, superimpose plots of these last three matrices using connected segments.

Then repeat the above sequence with half the step size and double the number of steps until the plots agree everywhere they share the same x values. Now try to guess then verify the exact solution if you have not already for the TAYLOR_ODES exercise.

DIRECTION_FIELD(r, x, x_0, x_m, m, y, y_0, y_n, n) approximates to a matrix of 2-component vectors that when plotted displays as a direction field for the equation $y' = r(x,y)$. x varies from x_0 through x_m in m steps, and y varies from y_0 through y_n in n steps. Before plotting, use the 2D-plot window's **Option State** command to connect the ends of the tangent segments and to make their endpoints small rather than large dots.

Exercise: Plot a direction field for the equation $y' = y - x$, by approximating the expression

```
DIRECTION_FIELD(y - x, x, -3, 3, 12, y, -2, 2, 4)
```

Switch to a 2D-Plot window and use the **Options State** command to connect segments and use small dots. Also use the **Options Color Auto** command and select No to make all the line segments the same color. Then plot the direction field.

Higher Order Differential Equations

A second or higher order ordinary differential equation can always be transformed into an equivalent system of first order differential equations as follows:

- First, introduce a unique new variable for each derivative except the highest.
- Next, replace the highest order derivative with the first derivative of the variable that represents the next-highest order derivative.
- Finally, for each of the other variables, add to the system an equation that equates its first derivative to the variable that represents the next higher order derivative.

For example, this procedure transforms a second order equation of the form $f(x, y'', y', y) = 0$ into the system of first order equations $f(x, v', v, y) = 0$ and $y' = v$.

Exercise: Transform the equation $y'' = \mathrm{SIN}(x + y^2 + y')$ into a system of two first order equations.

This procedure can also be used to transform a *system* of higher order ordinary differential equations into an equivalent system of first order equations. Simply transform each equation in turn, consistently using the same variable for a derivative everywhere it occurs.

Exercise: Transform the system

$$y'' = \mathrm{SIN}(x + y^2 + y' + z)$$

$$z'' = x \cdot y \cdot z + y' \cdot z + z'$$

into a system of four first order equations.

After transforming a higher order equation or system of equations into a system of first order equations, the methods described earlier in this section can be used to solve the system.

9.9 RECUREQN.MTH — Recurrence Equations

The file RECUREQN.MTH defines functions for finding exact solutions of first and second order difference equations and of linear-geometric recurrence equations.

LIN1_DIFFERENCE(p, q, x, x_0, y_0) simplifies to a specific solution of a recurrence equation having the form

$$y(x+1) = p(x)\cdot y(x) + q(x)$$

for the initial condition $y=y_0$ at $x=x_0$. Note that linear first order difference equations can always be put in this form. For a general instead of a specific solution, make x_0 and y_0 unassigned variables. A convenient way to do this is to give LIN1_DIFFERENCE only three arguments.

For example, to solve the difference equation $y(m+1) = m\cdot y(m) + (m+1)!$ for the initial condition $y=2$ at $m=1$, enter and simplify the expression

```
LIN1_DIFFERENCE(m, (m+1)!, m, 1, 2)
```

to produce the algebraic solution

$$\left[\frac{m^2}{2} + \frac{m}{2} + 1\right]\cdot(m-1)!$$

To verify that it satisfies the initial condition, substitute $m=1$ into the solution, and see that it simplifies to 2. To verify that it satisfies the difference equation $y(m+1) = m\cdot y(m) + (m+1)!$, substitute $m+1$ for m in the above solution, and see that that result equals m times the solution plus $(m+1)!$. Note that the best way to check that two dissimilar looking expressions are actually equal is to subtract one from the other and see if the result is 0.

The result of LIN1_DIFFERENCE may contain products or sums that ***DERIVE*** could not simplify to a closed-form. A closed-form solution for all $x > x_0$ may not even exist in terms of ***DERIVE***'s pre-defined functions. However:

RECURRENCE1(r, x, y, x_0, y_0, n) simplifies to the matrix

$$[[x_0, y_0],\ [x_0+1, y(x_0+1)],\ \ldots,\ [x_0+n, y(x_0+n)]]$$

where y satisfies the (perhaps nonlinear) first order difference equation $y(x+1) = r(x, y(x))$. Here $y(x)$ should be represented by a variable y rather than a function in expression r. Depending on r and your goal, it may be more appropriate to approximate rather than simplify when using RECURRENCE1. Either way, you can then plot the resulting two-column matrix.

Exercise: Use RECURRENCE1 to compute the first nine recurrences of

$$y(x+1) = x/(1+y(x)^2)$$

for the initial condition $y=1$ at $x=0$, and plot the result.

GEOMETRIC1(k, p, q, x, x_0, y_0) simplifies to a solution of the first order **linear-geometric recurrence** equation $y(k \cdot x) = p(x) \cdot y(x) + q(x)$, where k is a particular number, such as 2 or 3. The complexity analysis of "divide and conquer" algorithms often entails a linear-geometric recurrence equation.

Exercise: Find and verify a solution of the recurrence $z(2 \cdot m) = 4 \cdot z(m) + 5$ for the initial condition $z=0$ at $m=1$.

CLAIRAUT_DIF(p, q, d, x, y, c) simplifies to an implicit algebraic general solution of the equation $p(y-x \cdot d) = q(d)$, where variable y represents $y(x)$ and variable d represents the **forward first difference** $y(x+1) - y(x)$. This solution is in terms of x and a symbolic arbitrary constant c. For example, to find a general solution of the difference equation

$$(y(x+1) - x \cdot (y(x+1) - y(x)))^2 = (y(x+1) - y(x))^2$$

enter and simplify the expression

```
CLAIRAUT_DIF((y - x·d)^2, d^2)
```

to give

$$c^2 \cdot x^2 - 2 \cdot c \cdot x \cdot y + y^2 - c^2 = 0$$

Exercise: Verify that the above result satisfies the given difference equation.

LIN2_CCF(p, q, r, x, c_1, c_2) simplifies to a general solution of the constant coefficient second order linear difference equation $y(x+2) + p \cdot y(x+1) + q \cdot y(x) = r(x)$, in terms of two symbolic constants c_1 and c_2. Note that the result may contain one or more sums for which ***DERIVE*** can not find a closed-form.

Exercise: Find and verify a general solution of the equation

$$y(x+2) - 4 \cdot y(x+1) + 3 \cdot y(x) = 3^x$$

LIN2_CCF_BV(p, q, r, x, x_0, y_0, x_2, y_2,) is similar to LIN2_CCF, but simplifies to a specific solution that satisfies the boundary conditions $y=y_0$ at $x=x_0$ and $y=y_2$ at $x=x_2$.

Exercise: Find and verify a specific solution of the equation in the previous exercise for the boundary conditions $y=1$ at $x=0$ and $y=3$ at $x=1$.

9.10 APPROX.MTH — Pade Rational Approximation

The file APPROX.MTH has a function that computes Pade rational approximations to an expression:

PADE(y, x, x_0, n, d) simplifies to the **Pade rational approximation** to expression y depending on variable x, centered at x_0, requesting a numerator of degree n and a denominator of degree d. y must be differentiable through order $n+d$. You must have either $n=d$ or $n=d-1$. If a different degree difference between the numerator and denominator is appropriate, approximate $y\,x^k$ for the appropriate specific k, then divide the result by x^k.

Pade approximations are often more efficient and have a larger region of convergence than Taylor series — particularly in the neighborhood of non-polynomial behavior in y, such as asymptotes.

Exercise: Simplify

```
VECTOR([x, PADE(ê^x, x, 0, n, n)], n, 3)
```

and

```
VECTOR([x, TAYLOR(ê^x, x, 0, 2·n)], n, 3)
```

then open two 2D-plot windows and superimpose plots of the resulting two matrices on two plots of $\hat{e}^x$ to compare their performance.

9.11 EXP_INT.MTH — Exponential, Log, Sine, and Cosine Integrals

The file EXP_INT.MTH defines functions for approximating exponential, logarithmic, sine, and cosine integrals. The definitions, nomenclature, and organization in this section through 9.18 follow Abramowitz and Stegun's Handbook of Mathematical Functions, Dover Publications, Inc. [1965].

Many of the functions in this file are defined in terms of definite integrals for which you must use the **approX** command to determine an approximate numerical result for numerical values of the arguments. In such cases the accuracy is usually close to the current working precision if you receive no Dubious accuracy warning message. As a check you can temporarily increase the precision by five digits and compare the values.

In other cases these definitions are truncated series for which you must choose a particular number of terms. **Asymptotic series** generally diverge but have an error less than the first omitted term, so you can start with one term and keep increasing the number of terms while the change is diminishing in magnitude. In contrast, for a convergent series you can keep increasing the number of terms until there is no change in the displayed approximation. The exercises suggest techniques that help determine the appropriate number of terms.

In some cases you can determine other truncated series by expanding an integrand into an integrable truncated series, then integrating symbolically.

This section lists classic integral and series definitions for ease of identification. However, some of the integrals or series are transformed in the files into mathematically equivalent ones that are faster and more accurate to approximate.

EI(x, m) approximates to m terms of a series for the **exponential integral** Ei(x) defined for $x>0$ as the Cauchy principal value of

$$\int_{-x}^{\infty} \frac{\hat{e}^{-t}}{t}\, dt$$

Exercise: Using a precision of 6 significant digits (see Section 3.8), approximate

```
VECTOR(VECTOR(EI(x, m), x, 4), m, 6, 15)
```

to study convergence of this series as a function of x and m.

LI(x, m) approximates to m terms of a series for the **logarithmic integral** li(x) defined for $x > 1$ as the Cauchy principal value of

$$\int_0^x \frac{1}{\text{LN}(t)}\, dt$$

EN(n, z) approximates to the nth **exponential integral** $E_n(z)$ defined for the real part of $z > 0$ and n a nonnegative integer as

$$\int_1^\infty \frac{\hat{e}^{-z \cdot t}}{t^n}\, dt$$

EN_ASYMP(n, z, m) approximates to $m+1$ terms of an asymptotic series that approximates the above nth exponential integral for large magnitude z.

E1(z, m) approximates to m terms of a series for the 1st exponential integral $E_1(z)$ with |phase z| < π.

SI(z) approximates to the **sine integral** Si(z):

$$\int_0^z \frac{\text{SIN}(t)}{t}\, dt$$

Exercise: Define

$$\text{SI\_SERIES}(z, n) := \int_0^z \text{TAYLOR}\left[\frac{\text{SIN}(t)}{t}, t, 0, n\right] dt$$

then simplify SI_SERIES(z, 4).

CI(z) approximates to the **cosine integral** Ci(z) with |phase z| < π:

$$\text{euler\_gamma} + \text{LN}(z) + \int_0^z \frac{\text{COS}(t) - 1}{t}\, dt$$

9.12 PROBABIL.MTH — Probability Functions

The file PROBABIL.MTH defines the incomplete gamma function and its relatives, including various probability densities and cumulative distributions. Gamma and probability functions are described in Chapters 6 and 26 respectively of Abramonwitz and Stegun [1965]. See the introduction to Section 9.11 for a discussion of accuracy.

POCHHAMMER(a, x) simplifies to $(a)_x$, the Pochhammer's symbol function. $(a)_x$ is defined by the identity

$$(a)_x = \Gamma(a+x)/\Gamma(a)$$

PSI(z) approximates to $\psi(z)$, the psi function, provided the real part of z is positive (i.e. the phase of z is less than $\pi/2$). $\psi(z)$ is also called the digamma function. $\psi(z)$ is defined by the identity

$$\psi(z) = \frac{d}{dz} \ln \Gamma(z) = \frac{d}{dz} \Gamma(z) / \Gamma(z)$$

POLYGAMMA(n, z, m) simplifies to $m+1$ terms of a series approximation for $\psi^{(n)}(z)$, the nth polygamma function, provided z is not zero or a negative integer. $\psi^{(1)}(z)$, $\psi^{(2)}(z)$, $\psi^{(3)}(z)$, $\psi^{(4)}(z)$ are called the trigamma, tetragamma, pentagamma, and hexagamma functions respectively. $\psi^{(n)}(z)$ is defined by the identity

$$\psi^{(n)}(z) = \frac{d^n}{dz^n} \psi(z) = \frac{d^{n+1}}{dz^{n+1}} \ln \Gamma(z)$$

INCOMPLETE_GAMMA(z, w) approximates to $P(z, w)$, the incomplete gamma function, provided the real part of z is positive. $P(z, w)$ is defined as

$$\frac{1}{\Gamma(z)} \cdot \int_0^w \hat{e}^{-t} \cdot t^{z-1} \, dt$$

INCOMPLETE_GAMMA_SERIES(z, w, m) simplifies to $m+1$ terms of a series approximation for $P(z, w)$, the incomplete gamma function.

BETA(z, w) simplifies to $B(z, w)$, the beta function. $B(z, w)$ is defined by the identity

$$B(z, w) = \Gamma(z) \cdot \Gamma(w) / \Gamma(z+w)$$

INCOMPLETE_BETA(x, z, w) simplifies to $B_x(z, w)$, the incomplete beta function. $B_x(z, w)$ is defined as

$$\int_0^x t^{z-1} \cdot (1-t)^{w-1}\, dt$$

POISSON_DENSITY(k, t) simplifies to the Poisson probability density defined as

$$\frac{\hat{e}^{-t} \cdot t^k}{k!}$$

POISSON_DISTRIBUTION(k, t) simplifies to the cumulative probability Poisson distribution function defined as

$$\sum_{j=0}^{k} \frac{\hat{e}^{-t} \cdot t^j}{j!}$$

BINOMIAL_DENSITY(k, n, p) simplifies to the binomial probability density defined as

$$\text{COMB}(n, k) \cdot p^k \cdot (1-p)^{n-k}$$

BINOMIAL_DISTRIBUTION(k, n, p) simplifies to the cumulative probability binomial distribution function defined as

$$\sum_{j=0}^{\text{MIN}(k,n)} \text{BINOMIAL\_DENSITY}(j, n, p)$$

HYPERGEOMETRIC_DENSITY(k, n, m, j) simplifies to the hypergeometric probability density defined as

$$\frac{\text{COMB}(m, k) \cdot \text{COMB}(j-m, n-k)}{\text{COMB}(j, n)}$$

HYPERGEOMETRIC_DISTRIBUTION(k, n, m, j) simplifies to the cumulative probability hypergeometric distribution function defined as

$$\sum_{\alpha=\text{MAX}(0,n-j+m)}^{\text{MIN}(k,n,m)} \text{HYPERGEOMETRIC\_DENSITY}(\alpha, n, m, j)$$

STUDENT(t, v) approximates to $A(t|v)$, the cumulative probability student's t-distribution function. $A(t|v)$ is defined as

$$\frac{1}{\sqrt{v}\cdot \mathrm{BETA}\left[\frac{1}{2}, \frac{v}{2}\right]}\cdot\int_{-t}^{t}\left[1+\frac{x^2}{v}\right]^{-(v+1)/2} dx$$

F_DISTRIBUTION(F, v_1, v_2) approximates to $P(F|v_1, v_2)$, the cumulative probability F-distribution function. $P(F|v_1, v_2)$ is also called the variance-ratio distribution function. $P(F|v_1, v_2)$ is defined as

$$\frac{v1^{v1/2}\cdot v2^{v2/2}}{\mathrm{BETA}\left[\frac{v1}{2}, \frac{v2}{2}\right]}\cdot\int_{0}^{f} t^{(v1-2)/2}\cdot(v2+v1\cdot t)^{-(v1+v2)/2} dt$$

CHI_SQUARE(u, v) approximates to $P(\chi^2|v)$, the cumulative probability chi-square distribution function, where $u=\chi^2$. $P(\chi^2|v)$ is defined as

$$\frac{1}{2^{v/2}\cdot\Gamma\left[\frac{v}{2}\right]}\cdot\int_{0}^{u}\frac{t^{v/2-1}\cdot\hat{e}^{-t}}{2} dt$$

9.13 FRESNEL.MTH — Fresnel Integrals

The file FRESNEL.MTH defines functions for approximating Fresnel and related integrals. These integrals are described in Chapter 7 of Abramonwitz and Stegun [1965]. See the introduction to Section 9.11 for a discussion of accuracy.

DAWSON(x) simplifies to Dawson's integral $F(x)$ defined in terms of the complex error function (see Section 6.11):

$$-\frac{\sqrt{\pi}}{2}\cdot\hat{\imath}\cdot\hat{e}^{-x^2}\cdot\mathrm{ERF}(\hat{\imath}\cdot x)$$

FRESNEL_SIN(z) approximates to the Fresnel sine integral $S(z)$:

$$\int_0^z \mathrm{SIN}\left[\frac{\pi\cdot t^2}{2}\right]\,dt$$

FRESNEL_SIN_SERIES(z, m) approximates to $m+1$ terms of a series for the above Fresnel sine integral.

FRESNEL_SIN_ASYMP(z) simplifies to a five-term asymptotic approximation of the above Fresnel sine integral for large magnitude z.

FRESNEL_COS(z) approximates to the Fresnel cosine integral $C(z)$:

$$\int_0^z \mathrm{COS}\left[\frac{\pi\cdot t^2}{2}\right]\,dt$$

FRESNEL_COS_SERIES(z, m) approximates $m+1$ terms of a series for the above Fresnel cosine integral.

FRESNEL_COS_ASYMP(z) simplifies to a five-term asymptotic approximation of the above Fresnel cosine integral for large magnitude z.

9.14 BESSEL.MTH — Bessel and Airy Functions

The file BESSEL.MTH defines functions for approximating Bessel and Airy functions. These functions are described in Chapters 9 and 10 of Abramonwitz and Stegun [1965]. See the introduction to Section 9.11 for a discussion of accuracy.

BESSEL_J(n, z) approximates to the Bessel function of the first kind $J_n(z)$ for integer or fractional n. BESSEL_J is defined as

$$\frac{\left[\frac{z}{2}\right]^n}{\sqrt{\pi}\cdot\Gamma\left[n+\frac{1}{2}\right]}\cdot\int_0^{\pi}\mathrm{COS}(z\cdot\mathrm{COS}(t))\cdot\mathrm{SIN}(t)^{2\cdot n}\,dt$$

BESSEL_J_SERIES(n, z, m) approximates to $m+1$ terms of a series approximation for the Bessel function of the first kind $J_n(z)$.

BESSEL_J_ASYMP(n, z) simplifies to a one-term asymptotic approximation for the Bessel function of the first kind $J_n(z)$ for large magnitude z.

BESSEL_Y(n, z) approximates to the Bessel function of the second kind $Y_n(z)$ for noninteger n. BESSEL_Y is defined as

$$\frac{\mathrm{BESSEL\_J(n,\ z)}\cdot\mathrm{COS}(n\cdot\pi)-\mathrm{BESSEL\_J(\_n,\ z)}}{\mathrm{SIN}(n\cdot\pi)}$$

BESSEL_Y_SERIES(n, z, m) approximates to $m+1$ terms of a series approximation for the Bessel function of the second kind $Y_n(z)$ for integer n.

BESSEL_Y_ASYMP(n, z) simplifies to a one-term asymptotic approximation for the Bessel function of the second kind $Y_n(z)$ for large magnitude z.

BESSEL_I(n, z) approximates to the modified Bessel function of the first kind $I_n(z)$ for integer n. BESSEL_I is defined as

$$\frac{1}{\pi}\cdot\int_0^{\pi}\hat{e}^{z\cdot\mathrm{COS}(t)}\cdot\mathrm{COS}(n\cdot t)\,dt$$

BESSEL_I_SERIES(n, z, m) approximates to $m+1$ terms of a series approximation for the modified Bessel function of the first kind $I_n(z)$ for complex n.

BESSEL_I_ASYMP(n, z) simplifies to a two-term asymptotic approximation for the modified Bessel function of the first kind $I_n(z)$ for complex n and large magnitude z.

BESSEL_K(n, z) approximates to the modified Bessel function $K_n(z)$ for complex n and $|\text{phase } z| < \pi/2$. BESSEL_K is defined as

$$\int_0^\infty \hat{e}^{-z \cdot \text{COSH}(t)} \cdot \text{COSH}(n \cdot t)\ dt$$

BESSEL_K_ASYMP(n, z) simplifies to a two-term asymptotic approximation for the modified Bessel function $K_n(z)$ for complex n and large magnitude z.

SPHERICAL_BESSEL_J(n, z) simplifies to a closed form for the spherical Bessel function of the first kind $j_n(z)$ for integer n.

SPHERICAL_BESSEL_Y(n, z) simplifies to a closed form for the spherical Bessel function of the second kind, $y_n(z)$ for integer n.

AI_SERIES(z, m) approximates to $m+1$ terms of a series approximation for the Airy function Ai(z).

BI_SERIES(z, m) approximates to $m+1$ terms of a series approximation for the Airy function Bi(z).

9.15 HYPERGEO.MTH — Hypergeometric Functions

The file HYPERGEO.MTH defines functions for approximating hypergeometric functions. These functions are described in Chapters 13 and 15 of Abramonwitz and Stegun [1965]. See the introduction to Section 9.11 for a discussion of accuracy.

KUMMER(a, b, z) approximates to Kummer's **confluent hypergeometric function** $M(a, b, z) = {}_1F_1(a; b; z)$:

$$\frac{\Gamma(b)}{\Gamma(a)\cdot\Gamma(b-a)}\cdot\int_0^1 \hat{e}^{z\cdot t}\cdot t^{a-1}\cdot(1-t)^{b-a-1}\,dt$$

KUMMER_SERIES(a, b, z, m) approximates to $m+1$ terms of a series for the above confluent hypergeometric function.

HYPERGEOMETRIC(a, b, c, z) approximates to the **Gauss hypergeometric function** $F(a, b; c; z) + {}_2F_1(a, b; c; z)$:

$$\frac{\Gamma(c)}{\Gamma(b)\cdot\Gamma(c-b)}\cdot\int_0^1 t^{b-1}\cdot(1-t)^{c-b-1}\cdot(1-t\cdot z)^{-a}\,dt$$

HYPERGEOMETRIC_SERIES(a, b, c, z, m) approximates to $m+1$ terms of a series for the above Gauss hypergeometric function.

9.16 ELLIPTIC.MTH — Elliptic Integrals

The file ELLIPTIC.MTH defines functions for approximating elliptic integrals. These integrals are described in Chapter 17 of Abramonwitz and Stegun [1965]. See the introduction to Section 9.11 for a discussion of accuracy.

ELLIPTIC_F(Φ, m) approximates to the elliptic integral of the first kind $F(\Phi|m)$:

$$\int_0^{\Phi} \frac{1}{\sqrt{(1 - m \cdot \text{SIN}(t)^2)}}\, dt$$

ELLIPTIC_E(Φ, m) approximates to the elliptic integral of the second kind $E(\Phi|m)$:

$$\int_0^{\Phi} \sqrt{(1 - m \cdot \text{SIN}(t)^2)}\, dt$$

ELLIPTIC_PI(Φ, m, n) approximates to the elliptic integral of the third kind $\Pi(n;\Phi|m)$:

$$\int_0^{\Phi} \frac{1}{(1 - n \cdot \text{SIN}(t)^2) \cdot \sqrt{(1 - m \cdot \text{SIN}(t)^2)}}\, dt$$

9.17 ORTH_POL.MTH — Orthogonal Polynomials

The file ORTH_POL.MTH defines functions for generating orthogonal polynomials. These polynomials are described in Chapter 22 of Abramonwitz and Stegun [1965].

CHEBYCHEV_T(n, x) simplifies to the nth Chebychev polynomial of the first kind $T_n(x)$:

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_{n+1}(x) = 2 \cdot x \cdot T_n(x) - T_{n-1}(x)$$

Exercise: Simplify

```
VECTOR(CHEBYCHEV_T(n, t), n, 1, 8)
```

then plot the result from parameter value t=–1 to 1.

CHEBYCHEV_U(n, x) simplifies to the nth Chebychev polynomial of the second kind $U_n(x)$:

$$U_0(x) = 1$$

$$U_1(x) = 2\,x$$

$$U_{n+1}(x) = 2 \cdot x \cdot U_n(x) - U_{n-1}(x)$$

LEGENDRE_P(n, x) simplifies to the nth Legendre polynomial $P_n(x)$:

$$P_0(x) = 1$$

$$P_1(x) = x$$

$$(n+1) \cdot P_{n+1}(x) = (2 \cdot n+1) \cdot x \cdot P_n(x) - n \cdot P_{n-1}(x)$$

Exercise: To check orthogonality, verify that

$$\text{VECTOR}\left[\text{VECTOR}\left[\int_{-1}^{1} \text{LEGENDRE\_P(i)} \cdot \text{LEGENDRE\_P(j)}\ dx, j, 0, 2\right], i, 0, 2\right]$$

simplifies to a diagonal matrix.

ASSOCIATED_LEGENDRE_P(n, m, x) simplifies to the associated Legendre polynomial $P_n^m(x)$. ASSOCIATED_LEGENDRE_P is defined as

$$(-1)^m \cdot (1 - x^2)^{m/2} \cdot \left[\frac{d}{dx}\right]^m \text{LEGENDRE\_P}(x)$$

HERMITE_H(n, x) simplifies to the nth Hermite polynomial $H_n(x)$:

$$H_0(x) = 1$$

$$H_1(x) = 2\,x$$

$$H_{n+1}(x) = 2 \cdot x \cdot H_n(x) - 2 \cdot n \cdot H_{n-1}(x)$$

HERMITE_HE(n, x) simplifies to the nth associated Hermite polynomial $He_n(x)$:

$$He_0(x) = 1$$

$$He_1(x) = x$$

$$He_{n+1}(x) = x \cdot He_n(x) - n \cdot He_{n-1}(x)$$

WEBER_D(n, x) simplifies to Weber's nth parabolic cylinder function $D_n(x)$:

$$\hat{e}^{-x^2/4} \cdot \text{HERMITE\_HE}[n,\ x]$$

GENERALIZED_LAGUERRE(n, α, x) simplifies to the nth generalized Laguerre polynomial $L_n^{(\alpha)}(x)$:

$$L_0^{(\alpha)}(x) = 1$$

$$L_1^{(\alpha)}(x) = -x + \alpha + 1$$

$$(n+1) \cdot L_{n+1}^{(\alpha)}(x) = (2 \cdot n + \alpha + 1 - x) \cdot L_n^{(\alpha)}(x) - (n+\alpha) \cdot L_{n-1}^{(\alpha)}(x)$$

For the ordinary Laguerre polynomials, use 0 for α.

JACOBI_P(n, α, ß, x) simplifies to the nth Jacobi polynomial $P_n^{(\alpha,\,ß)}(x)$:

$$\frac{\left[\frac{d}{dx}\right]^n (1 - x)^\alpha \cdot (1 + x)^{ß} \cdot (1 - x^2)^n}{(-2)^n \cdot n! \cdot (1 - x)^\alpha \cdot (1 + x)^{ß}}$$

GEGENBAUER_C(n, α, x) simplifies to the nth Gegenbauer ultraspherical polynomial $\bar{C}_n^{\alpha}(x)$:

$$\frac{\Gamma\left[\alpha + \frac{1}{2}\right] \cdot \Gamma(n + 2 \cdot \alpha)}{(-2)^n \cdot n! \cdot \Gamma(2\alpha) \cdot \Gamma\left[\alpha + n + \frac{1}{2}\right] \cdot (1 - x^2)^{\alpha - 1/2}} \cdot \left[\frac{d}{dx}\right]^n (1 - x^2)^{n + \alpha - 1/2}$$

9.18 ZETA.MTH — Generalized Riemann Zeta Functions

The Riemann zeta function ZETA is a pre-defined function in ***DERIVE*** (see Section 6.11). The file ZETA.MTH defines several functions for approximating functions related to the zeta function. See the introduction to Section 9.11 for a discussion of accuracy.

HURWITZ_ZETA(s, a, m) approximates to $m+1$ terms of a series for the Hurwitz generalized zeta function:

$$\sum_{k=0}^{\infty} (k + a)^{-s}$$

POLYLOG(n, z, m) approximates to m terms of a series for Jonquière's polylogarithm function $\mathrm{Li}_n(z)$:

$$\sum_{k=1}^{\infty} \frac{z^k}{k^n}$$

LERCH_PHI(z, s, a, m) approximates to m terms of a series for the Lerch transcendent function $\Phi(z, s, a)$:

$$\sum_{k=0}^{\infty} \frac{z^k}{(a + k)^s}$$

DILOG(x) approximates to the dilogarithm function:

$$\int_1^x \frac{\mathrm{LN}(t)}{1 - t}\, dt$$

9.19 GRAPHICS.MTH Plotting Space Curves & Complex Expressions

The file GRAPHICS.MTH defines functions and constants that aid in the plotting of space curves, parametric surfaces, and expressions having complex values.

Space curves and parametric surfaces can be plotted in a 2D-plot window using an isometric projection, also called a trimetric projection: 3D images are projected perpendicular to the screen as if viewed from a great distance through a telephoto lens. For example, the isometric projection of the front surface of a cube does not appear larger (or smaller) than the back surface.

The 2D and 3D origins coincide. The positive 3D x-axis points down to the left and out of the screen. The positive 3D y-axis points symmetrically down to the right and out of the screen. The positive 3D z-axis points symmetrically up and out of the screen. This orientation is close to the default orientation in a 3D-plot window. However, hidden lines on parametric surfaces are not removed, so the effect is a wire frame rather than a scored opaque surface.

axes is assigned to a 3 x 2 matrix that parametrically specifies the isometric projections of the x, y, and z axes of the 3D Cartesian coordinate system. When **axes** is plotted in a 2D-plot window, you are successively prompted for the minimum and maximum values of the x-axis, the y-axis, and then the z-axis. These axes are drawn in that order from these minimum to maximum values. For example, use 0 as a minimum value if you only want to see the positive axis. Press **Ctrl-Enter** or **Ctrl-J** instead of **Enter** to make the same minimum and maximum values apply to the remaining axes.

Exercise: Split the ***DERIVE*** screen vertically into an algebra window and a 2D-plot window (see Section 2.13). In the algebra window load the file GRAPHICS.MTH, then enter and simplify **axes**. In the 2D-plot window, use the **Options Color Plot** command to make the axes and the cross the same color as the background color. (This makes the irrelevant 2D axes and cross invisible, thereby avoiding confusion while plotting space curves and parametric surfaces.) Finally plot **axes** with each axis going from a minimum value of 0 to a maximum value of 2. Note the order in which the positive 3D axes are drawn.

A 3D coordinate vector can represent a point, a space curve, or a parametric surface. The three elements of the vector respectively give the x, y, and z coordinates of the point, curve, or surface. If the coordinates are constants, the coordinate vector represents a point. If the coordinates depend on one parameter, the vector represents a space curve. If the coordinates depend on two parameters, the vector represents a parametric surface.

ISOMETRIC(v) simplifies to the 2D isometric projection of the 3D coordinate vector v representing a space curve.

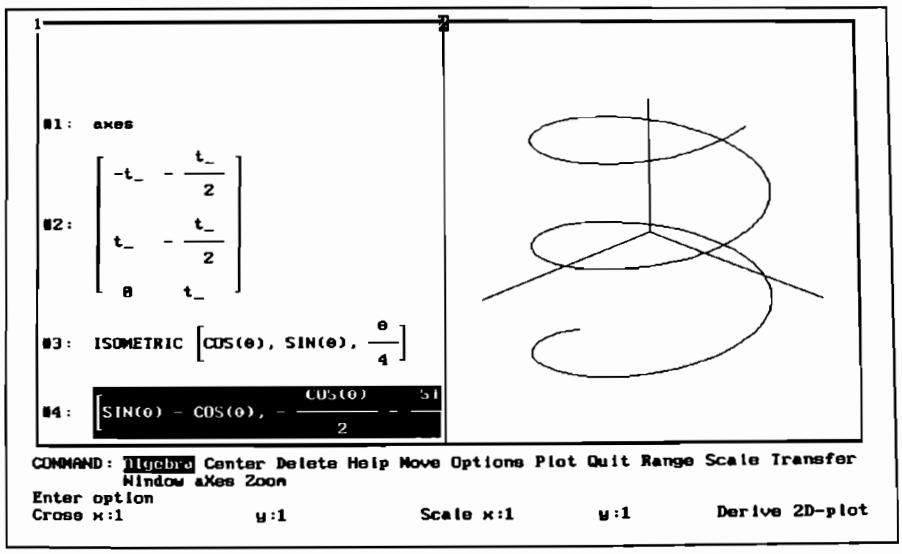
Exercise: A 3D coordinate vector for the helix space curve is

$$\left[\cos(\theta),\ \sin(\theta),\ \frac{\theta}{4}\right]$$

Determine the projection of the helix by simplifying the expression

```
ISOMETRIC([COS(θ), SIN(θ), θ/4])
```

Then plot the result for $\theta=-8$ to 8 in the 2D-plot window of the previous exercise. The screen should look similar to:


#1: axes
#2:
#3: ISOMETRIC
#4:
COMMAND: Algebra Center Delete Help Move Options Plot Quit Range Scale Transfer Window aXes Zoom
Enter option
Cross x:1 y:1 Scale x:1 y:1 Derive 2D-plot

A Space Curve Plot of a Helix

ROTATE_X(α) simplifies to a matrix A such that $A \cdot v$ rotates the 3D coordinate vector v through an angle of α radians about the x-axis, counterclockwise when viewed from the positive x-axis toward the origin.

ROTATE_Y(α) simplifies to a matrix A such that $A \cdot v$ rotates the 3D coordinate vector v through an angle of α radians about the y-axis, counterclockwise when viewed from the positive y-axis toward the origin.

ROTATE_Z(α) simplifies to a matrix A such that $A \cdot v$ rotates the 3D coordinate vector v through an angle of α radians about the z-axis, counterclockwise when viewed from the positive z-axis toward the origin.

Exercise: Use the **Delete Last** command to delete the plot of the helix from the 2D-plot window of the previous exercise. Rotate the helix coordinate vector 45 degrees ($\pi/4$ radians) about the x-axis by simplifying the expression

```
ROTATE_X(45·deg)·[COS(θ), SIN(θ), θ/4]
```

Use ISOMETRIC on the result to determine the isometric projection of this rotated helix coordinate vector, and then plot the projection. Similarly rotate and plot the original helix coordinate vector rotated 45 degrees about the y-axis, and then about the z-axis.

ISOMETRICS(v, s, s_0, s_m, m, t, t_0, t_n, n) approximates to a vector of $n+1$ two-column matrices that plots in a 2D-plot window as the isometric projection of the parametric surface specified by 3D coordinate vector v which is defined in terms of parameters s and t. This vector is evaluated on a grid of points with s varying from s_0 through s_m in m equal increments and t varying from t_0 through t_n in n equal increments.

To produce a plot of line segments along lines of constant t, use the 2D-plot window's **Options State** command to select the Rectangular coordinate system, Connected line segments, and Large points (see Section 5.1).

To make the dots and lines in such plots all the same color, use the **Options Color Auto** command to select No automatic color change.

COPROJECTION(A) approximates to the coprojection of A. The coprojection of a projection matrix produced by ISOMETRICS is a vector of $m+1$ two-column matrices that plots as the isometric projection of the same parametric surface, but with lines of constant s instead of constant t. Superimposing a plot of the coprojection onto a plot of the projection often helps visualize a 3-dimensional surface. Note that the coprojection can also be generated by approximating ISOMETRICS(v, t, t_0, t_n, n, s, s_0, s_m, m).

Note that in contrast to 3D-plot window plots, hidden lines are *not* removed from parametric surface plots. However, the parametric surfaces are *not* restricted to a Cartesian parameterization or to a single value of z for each pair of x-y coordinates.

SPHERE(r, Θ, Φ) simplifies to a 3D coordinate vector specifying the Cartesian coordinates of a point having spherical coordinates given by radius r, co-longitude Θ, and colatitude Φ. Θ is measured in radians counterclockwise in the x-y plane

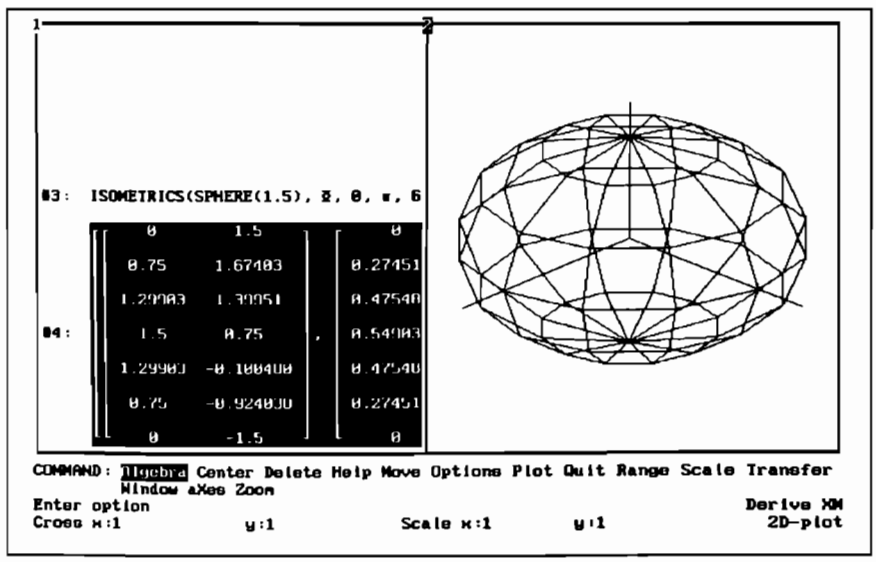
from the positive x-axis. Φ is measured in radians toward the x-y plane from the positive z-axis.

Beware that some authors define spherical coordinates differently, such as interchanging these definitions of Θ and Φ. To rotate or tilt the sphere, add appropriate constants to the angular parameters before simplifying. To shift the sphere away from the origin, add an appropriate 3D Cartesian coordinate vector to the result.

Exercise: Approximate

```
ISOMETRICS(SPHERE(1.5), Φ, 0, π, 6, θ, −π, π, 12)
```

to obtain a matrix representing an isometric projection of a sphere. Switch to a 2D-plot window and use the **Options Color Auto** and **Options State** commands to produce a single color plot with points connected by line segments. Then plot the sphere using lines of constant Φ. If you are using a color monitor, use the **Options Color Plot** command to change the color used for the next plot. Finally approximate and plot the coprojection of the matrix to superimpose lines of constant Θ. The screen should look similar to:



A Parametric Surface Plot of a Sphere

TORUS(r_c, r_s, Θ, Φ) simplifies to a 3D coordinate vector specifying the Cartesian coordinates of a point having toroidal coordinates r_s, Θ, and Φ, with r_c being the radius from the origin to the center of the tube. The circle that is the center of the

tube lies in the x-y plane, centered at the origin. r_s is the radius perpendicular to the center of the tube. Θ measures the angle projected onto the x-y plane, counter-clockwise from the positive x-axis. Φ measures the angle around the tube, up from the ray pointing from the center of the tube toward the origin.

Exercise: Approximate

```
ISOMETRICS(TORUS(6/5, 1/3), θ, -π, π, 12, Φ, -π, π, 6)
```

to obtain a matrix representing an isometric projection of a torus and plot the result. Then approximate and plot the coprojection of the matrix to superimpose lines of constant Θ.

CYLINDER(r, Θ, z) simplifies to a 3D coordinate vector specifying the Cartesian coordinates of a circular cylinder of radius r from the z-axis.

Exercise: Approximate

```
ISOMETRICS(CYLINDER(1.5), z, -5/4, 5/4, 1, θ, -π, π, 25)
```

to obtain a matrix representing an isometric projection of a cylinder and plot the result. Then approximate and plot the coprojection of the matrix to connect the rims at the ends of the cylinder.

CONE(α, Θ, z) simplifies to a 3D coordinate vector specifying the Cartesian coordinates of a circular cone having a surface at angle α from the z-axis, centered at the origin.

Exercise: Approximate

```
ISOMETRICS(CONE(π/4), z, -5/4, 5/4, 1, θ, -π, π, 16)
```

to obtain a matrix representing an isometric projection of a cone and plot the result. Then approximate and plot the coprojection of the matrix to connect the rims at the ends of the cone.

There are several ways to plot expressions having complex values. If the expression depends on only one real variable, 2D plots of the real and imaginary parts, or the absolute value and phase, can be superimposed. Alternatively discrete and continuous parametric plots of a vector of the real and imaginary parts can be superimposed, enabling you to estimate the value of the real variable along the curve.

Exercise: Start a fresh ***DERIVE*** session (see Section 2.1). Use the **Window Designate** command to make a full screen 2D-plot window. Use the **Window Split** commands to split the screen horizontally and then split each half vertically,

to produce four 2D-plot windows. Redesignate one of these windows to be an algebra window and enter the assignment

$$\text{expn} := \hat{e}^{\hat{i}\cdot t}\cdot \text{ATAN}(t)$$

Then superimpose plots of RE(*expn*) and IM(*expn*) in one 2D-plot window. Next superimpose plots of ABS(*expn*) and PHASE(*expn*) in another 2D-plot window. (Press the **F1** function key to switch between windows.) Finally superimpose continuous and discrete plots of the vector

$$[\text{RE}(\text{expn}),\ \text{IM}(\text{expn})]$$

in the third 2D-plot window using 20 points. Try to visualize the nature of *expn* using these three images.

If an expression having complex values depends on one complex variable z, $x+\hat{i}y$ can be substituted for z and then 3D plots of the real and imaginary parts can be made in separate 3D-plot windows. Alternatively you can plot the absolute value and the phase in separate 3D-plot windows.

Exercise: Start a fresh ***DERIVE*** session and split the screen into four algebra windows. In window number 1, enter the expression

$$\text{RE}(\hat{e}^{x + \hat{i}\cdot y})$$

Use the **Window Open** command to overlay a 3D-plot window over that algebra window. Then use the **Length** command to set the x length to 2 and the y length to 4π and plot the expression. Repeat this process in the other three windows for the expressions

$$\text{IM}(\hat{e}^{x + \hat{i}\cdot y})$$

$$\text{ABS}(\hat{e}^{x + \hat{i}\cdot y})$$

and

$$\text{PHASE}(\hat{e}^{x + \hat{i}\cdot y})$$

Try to visualize the nature of $\hat{e}^z$ using these four images.

Another way of plotting a complex-valued expression w depending on a complex variable z is to show in a 2D-plot window at each point on a rectangular grid of z coordinates a ray having the length and direction of the corresponding w value. You can mentally interpolate to estimate the polar w values between rectangular z grid points. You can also mentally project the rays into rectangular components.

RAYS(w, z, z_{00}, z_{mn}, m, n) approximates to an array that plots in a 2D-plot window as a rectangular grid of rays as described in the preceding paragraph. z varies from z_{00} at one corner of the grid through z_{mn} at the diagonally opposite corner, with m panels in the x direction and n panels in the y direction. Multiply w by a suitable numerical scale factor if the rays are too short to discern accurately or so long that they confusingly cross each other.

To have each ray drawn as a line segment joining two small dots, with a large dot superimposed at the origin of the ray, use the 2D-plot window's **Options State** command and select the Rectangular coordinate system, Connected line segments, and Small points.

Exercise: Approximate

```
RAYS(ê^z, z, -1 - 2·π·î, 1 + 2·π·î, 4, 8)
```

and plot the result in a 2D-plot window. Together with the 3D plots of the previous exercise, this 2D plot should reinforce your understanding of how $\hat{e}^z$ varies exponentially with the real part of z and sinusoidally with the imaginary part of z.

Another method of plotting a complex valued expression w depending on a complex variable z is to show how a rectangular grid of values in the z-plane maps into a distorted grid in the w-plane. The images of each row of z values can be joined by line segments to help visualize the underlying grid topology. By noting the order in which the points are drawn and counting them, each w-plane grid point can be associated with the corresponding z-plane coordinates. Then you can mentally interpolate to estimate the w values for intermediate z values.

HORIZONTALS(w, z, z_{00}, z_{mn}, m, n) approximates to a matrix that plots in a 2D-plot window as an array of dots, with each row of w values joined by segments. z varies from z_{00} at one corner of the grid through z_{mn} at the diagonally opposite corner, with m panels in the x direction and n panels in the y direction. Use the **Options State** command to connect dots and make them larger than the line segments. To join the dot columns by segments, approximate and plot the coprojection of the matrix.

Exercise: Approximate

```
HORIZONTALS(ê^z, z, -1 - π·î, 1 + π·î, 4, 12)
```

and plot the result in a 2D-plot window. Then approximate and plot the coprojection of the matrix to superimpose the plots.

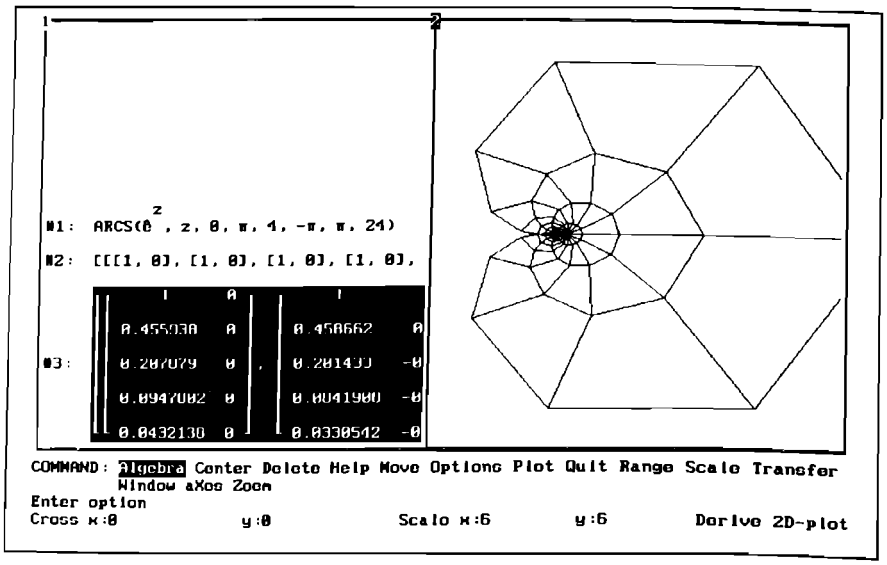
A close alternative to the preceding plotting technique is to use a regular polar array of z values, $z = r \cdot \hat{e}^{i\Theta}$. The corresponding dots in the w-plane can be joined along lines of constant radius and/or lines of constant phase in the z-plane to help visualize the grid topology.

ARCS(w, z, r_0, r_m, m, Θ_0, Θ_n, n) approximates to a matrix that plots in a 2D-plot window as an array of dots, with each row of w values joined by segments. $z = r \cdot \hat{e}^{i\Theta}$ varies from $r = r_0$ through r_m and $\Theta = \Theta_0$ through Θ_n, with m panels in the r direction and n panels in the Θ direction. To join the dots in the radial direction, approximate and plot the coprojection of the matrix.

Exercise: Approximate

$$\text{ARCS}(\hat{e}^z,\ z,\ 0,\ \pi,\ 4,\ -\pi,\ \pi,\ 24)$$

and plot the result in a 2D-plot window. Then approximate and plot the coprojection of the matrix to superimpose the plots. The screen should look similar to:


#1: ARCS(ê^z, z, 0, π, 4, -π, π, 24)
#2: [[[1, 0], [1, 0], [1, 0], [1, 0],
#3:
COMMAND: Algebra Center Delete Help Move Options Plot Quit Range Scale Transfer Window aXes Zoom
Enter option
Cross x:0 y:0 Scale x:6 y:6 Derive 2D-plot

A w-plane Image of a Polar Grid

9.20 NUMBER.MTH — Number Theory Functions

The exact precision integer and rational arithmetic provided by ***DERIVE*** make it ideal for explorations into the theory of numbers. The pre-defined piecewise continuous functions (see Section 6.7) and probability functions (see Section 6.9) are especially useful for this purpose. The file NUMBER.MTH defines additional number theory functions in terms of the pre-defined functions.

CEILING(m, n) simplifies to the smallest integer greater than or equal to m/n. n defaults to 1. If m or n are not numbers, CEILING(m, n) simplifies to the equivalent expression $-$ FLOOR($-$ m/n).

ROUND(m, n) simplifies to the nearest integer to m/n. n defaults to 1. If m or n are not numbers, ROUND(m, n) simplifies to the equivalent expression FLOOR($m/n+1/2$).

SQUARE_WAVE(n) simplifies to 1 if n mod 2 is less than 1, otherwise it simplifies to -1.

POWER_MOD(n, d, m) returns n^d mod m. For example, POWER_MOD(2, 500, 100) simplifies to 76. POWER_MOD is *no* more efficient than computing MOD(n^d, m) since MOD now efficiently computes n^d modulo m (see Section 6.7).

NTH_PRIME(n) simplifies to the nth prime number. For example, NTH_PRIME(3) simplifies to 5.

Exercise: Make a table of the first 50 primes by simplifying the expression

```
VECTOR(VECTOR(NTH_PRIME(n + m), m, 0, 40, 10), n, 1, 10)
```

CATALAN(n) simplifies to the nth Catalan number. For example, CATALAN(10) simplifies to 16796.

BERNOULLI(n) simplifies to the nth Bernoulli number. For example, BERNOULLI(10) simplifies to 5/66.

BERNOULLI_POLY(n, x) simplifies to the nth Bernoulli polynomial evaluated at x. For example, BERNOULLI_POLY(10, x) simplifies to

$$\frac{66\cdot x^{10} - 330\cdot x^{9} + 495\cdot x^{8} - 462\cdot x^{6} + 330\cdot x^{4} - 99\cdot x^{2} + 5}{66}$$

EULER(n) simplifies to the nth Euler number. For example, EULER(10) simplifies to −50521.

EULER_POLY(n, x) simplifies to the nth Euler polynomial evaluated at x. For example, EULER_POLY(10, x) simplifies to

$$x \cdot (x^9 - 5 \cdot x^8 + 30 \cdot x^6 - 126 \cdot x^4 + 255 \cdot x^2 - 155)$$

FIBONACCI(n) simplifies to the nth Fibonacci number. For example, FIBONACCI(10) simplifies to 55.

MERSENNE(n) simplifies to the nth Mersenne prime. For example, MERSENNE(10) simplifies to 618970019642690137449562111.

CONTINUED_FRACTION(u, n) simplifies or approximates to a vector of $n+1$ partial quotients of the continued fraction of u. For example, CONTINUED_FRACTION(#e,8) approximates to [2, 1, 2, 1, 1, 4, 1, 1, 6]. If a ? appears in the result, use the **Options Precision** command (see Section 3.8) to increase the digits of precision.

CONVERGENT(v) simplifies to the fraction corresponding to the vector v of partial quotients (such as those returned by CONTINUED_FRACTION). For example CONVERGENT([2, 1, 2, 1, 1, 4, 1, 1, 6]) simplifies to 1264/465, an approximation for #e.

PARTS(n) simplifies to the number of decompositions of n into integer summands without regard to order. For example, 4 = 1+3 = 2+2 = 1+1+2 = 1+1+1 so PARTS(4) simplifies to 5.

DISTINCT_PARTS(n) simplifies to the number of decompositions of n into distinct integer summands without regard to order. For example, 4 = 1+3 so DISTINCT_PARTS(4) simplifies to 2.

9.21 MISC.MTH — Miscellaneous Utility Functions

The file MISC.MTH contains functions for miscellaneous tasks that do not fit the categories of the other utility files.

Consider the infinite series

$$\sum_{n=a}^{\infty} t$$

where term t is an expression dependent on index variable n. To determine if the series converges or diverges for a given t, first enter the above sum and see if it simplifies to a closed form of finite or infinite magnitude. Success in finding a closed form sum may require restricting the domains of variables in t (other than the index variable n) to appropriate intervals (see Section 4.10).

If ***DERIVE*** does not return a closed form sum, there are many tests you can use to determine if a series converges or diverges. For example, Leibniz's alternating series test states that a series converges if

- t alternates in sign beyond a certain n,
- the limit of t as n approaches ∞ is 0, and
- $|t|$ is nonincreasing beyond a certain n.

Convergence tests for *non*-alternating series are given in Calculus textbooks. Most of these tests are easy to apply directly using ***DERIVE***'s **Calculus Limit** or **Calculus Integrate** commands. However, the particularly useful ratio test requires a substitution followed by a limit, so the following function is defined in MISC.MTH to facilitate performing this test:

RATIO_TEST(t, n) simplifies to the limit of the absolute value of the ratio of the $n+1$th term of the series to the nth term, as n approaches ∞. If the result is greater than 1, the series diverges. If the result is less than 1, the series converges. If the result equals 1, the test is inconclusive.

Exercise: Use RATIO_TEST to determine whether the series

$$\sum_{n=1}^{\infty} \frac{n!}{n^n}$$

converges or diverges.

If $u(x,y)$ is an expression dependent on the variables x and y, ***DERIVE*** simplifies LIM(u, [x, y], [x_0, y_0]) by first computing the limit with respect to x, then with respect to y. This result may be different than LIM(u, [y, x], [y_0, x_0]). In contrast, the following function returns the multivariate limit:

LIM2(u, x, y, x_0, y_0) simplifies to the limit of u as [x, y] approaches [x_0, y_0] along a straight straight line of slope @1. If the result is independent of @1, the limit is independent of the direction of approach.

Exercise: Use LIM2 to show that as x and y approach 0 the limit of

$$\frac{x^2 - y^2}{x^2 + y^2}$$

is *non*unique and that the limit of

$$\frac{\text{SIN}(x^2) - \text{SIN}(y^2)}{x - y}$$

is unique.

The adaptive Simpson's Rule method ***DERIVE*** uses to numerically approximate definite integrals is fast, robust, and accurate (see Section 7.4). However, simpler numerical integration methods are useful for educational purposes; for obtaining rigorous bounds on the integrals of monotonic, concave, or convex integrands; and for approximating definite integrals when the integrands depend on parameters other than the integration variable.

LEFT_RIEMANN(u, x, a, b, n) approximates to the left **Riemann sum** for the integral of expression u from $x=a$ through b, using n equal-width rectangular strips. If ***DERIVE*** can determine a closed form for the sum for symbolic n, then you may prefer to use **Simplify** rather than **approX** so that you can simplify the limit of that closed-form sum as n approaches ∞. Otherwise you should use numeric n and **approX**.

Exercise: Approximate

$$\int_0^1 \text{COS}(\text{SIN}(x))\ dx$$

Then for n = 16, 32, 64, ..., approximate

```
LEFT_RIEMANN(COS(SIN(x)), x, 0, 1, n)
```

to study how these left Riemann sums converge as n approaches infinity by successive doublings.

Exercise: For the right Riemann sum, u matches the right rather than the left edges of the rectangles. COS(SIN(x)) is monotonic on $0 \le x \le 1$, so to within roundoff, the integral is bounded between the left and right Riemann sums. Inspect the definition of LEFT_RIEMANN, then define an analogous function RIGHT_RIEMANN and study how it converges to this integral from the side opposite that in the previous exercise.

Exercise: Define an analogous function named MID_VALUES for which u matches the height at the middle of the rectangles, then study how much faster it converges for the above integral.

Exercise: Define an analogous function named TRAPEZOIDS for which the integral is approximated as the area of n trapezoids of equal width. Noting that the above integrand is concave downward throughout the interval, study how fast this method converges from the opposite side as MID_VALUES. Asymptotically what simple weighted average of the corresponding midvalue and trapezoidal results is particularly accurate?

There may be expressions for which ***DERIVE*** is unable to find a closed form antiderivative, even though one exists. You may be able to help ***DERIVE*** integrate such expressions by using one of the following functions to explicitly integrate by parts or to make a change of variable.

INT_PARTS(u, v, x) uses integration by parts to find the antiderivative of the expression $u \cdot v$ with respect to x by simplifying the equivalent expression

$$u \cdot \int v \, dx - \int \left(\int v \, dx \cdot \frac{d}{dx} u \right) dx$$

INT_SUBST(y, x, u) finds the antiderivative of the expression $y(x)$ with respect to x by substituting the inverse of $u(x)$ for x in y, integrating, and then substituting $u(x)$ for x in the result. This technique is called integration by substitution. For example,

```
INT_SUBST(t·SIN(t^2), t, t^2)
```

simplifies to

$$-\frac{\cos(t^2)}{2}$$

Exercise: Find expressions that ***DERIVE*** cannot directly integrate, but for which an antiderivative can be found using INT_PARTS or INT_SUBST. If you find any such expressions, send them to Soft Warehouse, Inc. so we can improve the system!

DEF_INT_SUBST(y, x, u, a, b) finds the definite integral of the expression $y(x)$ from $x=a$ to $x=b$ by substituting the inverse of $u(x)$ for x in y and integrating the result from $u(a)$ to $u(b)$.

INVERSE(u, x) simplifies to the inverse of expression $u(x)$ with respect to x. For example, INVERSE(SIN(x/b), x) simplifies to b ASIN(x). Note that making appropriate domain declarations (see Section 4.10) of variables that occur in u will help INVERSE produce the desired result.

DERIVE can assist with many proofs, including inductive ones. The classic example of an inductive proof is one that

$$\sum_{k=a}^{n} t = s$$

where t is an expression depending on k and s is an expression depending on n. This can be done by proving that $t(a) - s(a) = 0$ and proving that

$$s(n) + t(n+1) - s(n+1) = 0$$

This is just substitution followed by algebraic simplification, seeking two expressions equivalent to zero. Consequently, it is easily automated using the LIM function to do the substitutions:

PROVE_SUM(t, k, a, n, s) simplifies to a vector of two expressions. If both expressions are equivalent to 0, the sum formula above is proved. If either expression is not equivalent to 0, the sum formula is disproved.

Exercise: Use PROVE_SUM to prove or disprove that

$$\sum_{k=1}^{n} k! \cdot k = (n + 1)! - 1$$

The technique used in PROVE_SUM can be adapted to many other broad categories of inductive proofs.

Exercise: Define an analogous function PROVE_PRODUCT(*f*, *k*, *a*, *n*, *p*) that proves or disproves

$$\prod_{k=a}^{n} f = p$$

Exercise: Use PROVE_PRODUCT to prove or disprove that

$$\prod_{k=1}^{n} 2 \cdot k \cdot 3^{k} = 2^{n \cdot (n+1)/2} \cdot n! \cdot 3^{n}$$

The following two functions are for extracting components of polynomials:

POLY_COEFF(*u*, *x*, *n*) simplifies to the coefficient of the x^n term(s) of the polynomial $u(x)$.

Exercise: Use POLY_COEFF to find the coefficient of the x^3 terms of

$$3 \cdot x^4 - 5 \cdot a \cdot x^3 + 2 \cdot x^3 \cdot y^2 - 7 \cdot x \cdot y^3$$

POLY_DEGREE(*u*, *x*) simplifies to the degree of *x* in the polynomial $u(x)$. Note that it returns −1 as the degree of 0, the zero polynomial. If desired, the definition of POLY_DEGREE can be modified using an IF clause (see Section 10.3) to return 0 if $u=0$. Others may prefer that it be modified to return $-\infty$ if $u=0$.

Exercise: Use POLY_DEGREE to find the degree of *x* and the degree of *y* in the polynomial of the previous exercise.

The following functions and variable show how the pseudo-random number generator function (see Section 6.9) can be used:

random_sign simplifies to either 1 or −1 randomly with equal probability.

RANDOM_POLY(*x*, *d*, *s*) simplifies to a polynomial of degree *d* in the variable *x* with random number coefficients between −*s* and *s* exclusive.

RANDOM_VECTOR(*n*, *s*) simplifies to an *n* element vector with random number elements between −*s* and *s* exclusive.

RANDOM_MATRIX(*m*, *n*, *s*) simplifies to an *m* x *n* element matrix with random number elements between −*s* and *s* exclusive.

The following function indicates how well an expression approximates a set of data points:

GOODNESS_OF_FIT (u, x, A) returns the standard deviation of expression $u(x)$ from the values given in data matrix A. $u(x)$ can be the expression returned by the pre-defined function FIT, provided $u(x)$ is dependent on only one data variable (see Section 6.10). For numerical data, GOODNESS_OF_FIT should be approximated rather than simplified.

Exercise: Find the goodness of fit of the data to the result of the last example in Section 6.10.

POLY_INTERPOLATE (A, x) simplifies to the polynomial in variable x that interpolates the $[x, y]$ coordinate pairs given by the rows of the two-column matrix A. When applicable, POLY_INTERPOLATE is generally preferable to the pre-defined FIT function (see Section 6.10). For example,

```
POLY_INTERPOLATE([0,0], [1,1], [2,2], [3,0]], x)
```

simplifies to

$$\frac{x^2 \cdot (3 - x)}{2}$$

POLY_INTERPOLATE_EXPRESSION (u, x, a) simplifies to the polynomial in variable x that interpolates the expression u given the one dimensional vector of supporting points a.

Chapter 10. Programming

In ***DERIVE***, a program is just a MTH file of related function definitions and variable assignments. The utility files distributed with ***DERIVE*** are examples of such programs (see Chapter 9). To use a program, load the MTH file into an algebra window using the **Transfer Load Utility** command (see Section 2.9). Once loaded, the functions and variables defined in the MTH file can be used in mathematical expressions in the same way that the pre-defined functions and constants are used.

Section 4.11 describes how to assign values to variables. Section 4.12 describes how to define functions. This chapter describes how to write your own programs using the various tools provided by ***DERIVE***.

10.1 The ITERATES Function

In most programming languages numerical operations are usually done by iteration (i.e. loops and jumps) and/or by recursion (i.e. recursive function calls). Many of these operations can easily be done in ***DERIVE*** using its pre-defined calculus and vector functions (see Chapter 7 and 8). For example, SUM and PRODUCT can be used to add or multiply a sequence of expressions. VECTOR can be used to generate a table of data. The SUB operator can be used to extract data from a vector or table.

In addition to these specialized functions, ***DERIVE*** has two general-purpose iteration functions. For example, suppose you want to solve the transcendental equation

$$x = \hat{e}^{-x/20}$$

A plot of the difference in the two sides of the equation suggests a solution in the neighborhood of $x=1$. To refine this initial guess, substitute the guess into the right side of the equation, and then use the **approX** command on the result to determine a second approximation for x. Then substitute this second approximation for x in the right side to determine a third approximation, and so on.

The **approX** command rounds to the current precision, so perhaps eventually one of these approximate x values will round to one of the earlier ones. From then on the sequence of approximations between those two would repeat endlessly, so it is pointless to iterate further without increasing the precision.

The pre-defined function ITERATES can automate this fixed-point iteration method for solving equations. If you approximate expressions of the form

ITERATES (u, x, x_0)

the update formula $x \leftarrow u(x)$ is repeatedly applied starting with $x=x_0$ until x is equal to one of its earlier values. The result is a vector of these values, starting with x_0. In exact mode, the **Simplify** command would never yield an x equal to an earlier one, so the **approX** command is appropriate for this application of ITERATES. For example,

```
ITERATES(ê^(-x/20), x, 1)
```

approximates to the vector of decimal numbers: [1, 0.951229, 0.953551, 0.953441, 0.953446, 0.953446, 0.953446, 0.953446].

While it appears that the last *four* values are equal, simplifying and displaying this vector in rational notation gives

$$\left[1, \frac{4681}{4921}, \frac{349}{366}, \frac{8437}{8849}, \frac{19559}{20514}, \frac{21013}{22039}, \frac{18412}{19311}, \frac{18412}{19311}\right]$$

Thus only the last *two* values are actually equal. The decimal notation only makes it *appear* that ITERATES gave more iterates than necessary.

ITERATES takes an optional fourth argument giving exactly how many iterations to make. For example,

```
ITERATES(ê^(-x/20), x, 1, 5)
```

approximates to the six element vector: [1, 0.951229, 0.953551, 0.953441, 0.953446, 0.953446].

If the fourth argument of ITERATES is a negative integer, the inverse of the first argument is iterated upon. For example,

```
ITERATES(TAN(x), x, x, -1)
```

simplifies to [x, ATAN(x)]. This feature is used to define INVERSE in MISC.MTH (see Section 9.21).

The fixed-point iteration method often diverges since the magnitudes of successive iterates get bigger and bigger until memory is exhausted. Thus, it is usually a good idea to make the fourth argument of ITERATES a small positive integer. Section 9.1 discusses fixed-point iteration in more detail.

Newton's method for solving an equation provides another good illustration of the use of ITERATES. To solve equations of the form $u=0$ with an initial guess of $x=x_0$, define NEWTON as follows:

```
NEWTON(u, x, x0, n) := ITERATES(x - u/DIF(u, x), x, x0, n)
```

Then NEWTON applies the update formula

$$x \leftarrow x - \frac{u}{du/dx}$$

to correct the current approximation for x by subtracting u divided by the derivative of u and evaluating the result at x. If n is a positive integer, the iteration is repeated n times; otherwise, it is repeated until x is equal to one of its earlier values. Section 9.1 discusses Newton's method in more detail.

Exercise: Enter the above definition of NEWTON, and use the **Transfer Save Derive** command to save it in the file NEWTON.MTH (see Section 2.8). Next, solve the equation $x^2 = 3$ by approximating the expression

```
NEWTON(x^2 - 3, x, 2)
```

The last iterate should be an approximation to the square root of 3. Similarly use NEWTON to approximate the fifth root of 100.

Exercise: Find the complex solutions of the equation $\hat{e}^z - z^2 = 0$ by approximating

```
NEWTON(ê^z - z^2, z, z0, 9)
```

after substituting for z_0 appropriate complex numbers as initial guesses.

> **Hint**: To determine initial guesses, substitute the symbolic expression $x+y\cdot\hat{i}$ for z in $\hat{e}^z - z^2$ and simplify, giving an expression u. Then plot $-|\mathrm{RE}(u)| - |\mathrm{IM}(u)|$ in a 3D-plot window to see where it, and hence u, is approximately 0.

Exercise: To determine a sequence of rational approximations to $\sqrt{a}$ centered at $a=1$, simplify (do not approximate)

```
NEWTON(x^2 - a, x, 1, 3)
```

Then superimpose plots of each iterate on a plot of $\sqrt{a}$ to compare their accuracies in the neighborhood of $a=1$.

As another example of the use of ITERATES, consider the continued fraction

$$b + \cfrac{a}{b + \cfrac{a}{\cdots + \cfrac{a}{b + \cfrac{a}{b + \cfrac{a}{x}}}}}$$

where there are n divisions. Entering such expressions on the author line would be tedious, especially for large n. However, it is possible to generate the continued fraction by applying the update formula $r \leftarrow b+a/r$ to itself n times, starting with $r=x$.

The following definition of CF uses ITERATES to generate a vector of continued fractions:

```
CF(b, a, x, n) := ITERATES(b + a/r, r, x, n)
```

Exercise: Enter the above definition for CF and then trivially factor CF(1, 1, x, 6). From these results, try to predict the form for larger n.

Note that a function for computing a vector of the partial quotients of an expression's continued fraction is defined in NUMBER.MTH (see Section 9.20).

10.2 The ITERATE Function

In many cases only the last iterate is of interest, particularly when the fourth argument is omitted or the **Simplify** command is used so that convergence is less of an issue. ITERATE is identical to ITERATES, except that ITERATE returns only the last iterate. ITERATE returns ? if the iteration does not converge to a single unique value (i.e. if the cycle length is *not* 1).

For example, to compute the nth power of x, define the function

```
POWER(x, n) := ITERATE(a·x, a, 1, n)
```

Here the update formula $a \leftarrow a\,x$ is being applied n times, starting the accumulator a with the initial value 1. This iteration multiplies n copies of x, giving x^n. Although ***DERIVE*** has a built-in ^ operator for raising an expression to a power, POWER is a good elementary application demonstrating the use of ITERATE.

Exercise: Define the function POWER using the above definition. Use it to compute 2^0, 2^1, 2^3, and 2^{100}.

Exercise: Define the function NTH_PRIME(n) to return the nth prime number. For example, NTH_PRIME(3) should return 5. Use it to verify that the 1000th prime number is 7919.

> **Hint**: Use the function NEXT_PRIME (see Section 6.7). Note that NTH_PRIME is also defined in NUMBER.MTH (see Section 9.20).

When an iteration depends on two or more of the previous iterates, you can use a vector with ITERATE to store the previous iterates. For example, the Fibonacci numbers are defined by the recurrence

$$\begin{aligned} \text{FIB}(n) &= 0 && \text{if } n = 0, \\ \text{FIB}(n) &= 1 && \text{if } n = 1, \text{ and} \\ \text{FIB}(n) &= \text{FIB}(n-1) + \text{FIB}(n-2) && \text{if } n > 1. \end{aligned}$$

where n is a nonnegative integer. Use a two-element vector v to store the value of FIB(n–2) and FIB(n–1). Then by letting n=2 the above recurrence specifies that the initial value of v is [0, 1]. Also it specifies that to update v, make its new first component its previous second component and make its new second component the sum of the two previous components. This initialization and update are accomplished using ITERATE as follows:

```
FIB(n) := ITERATE([v SUB 2, v SUB 1 + v SUB 2], v, [0, 1], n)
```

If FIB is defined as shown and n is a nonnegative integer, FIB(n) will simplify to a vector whose first element is the nth Fibonacci number and whose second is the $n+1$th Fibonacci number. For example, FIB(100) simplifies to

```
[354224848179261915075, 573147844013817084101]
```

and 354224848179261915075 is the 100th Fibonacci number.

Exercise: Use the SUB operator to modify the definition of FIB so only the first element of the vector is returned. Verify that your modified FIB gives the correct result for FIB(100).

Instead of a variable, the second argument of ITERATE and ITERATES can be a vector of variables and the third argument a vector of values for those variables. This can simplify definitions using these functions by eliminating the need to use subscripts. For example, this simplifes the definition for FIB given earlier to

```
FIB (n) := ITERATE([k, j+k], [j, k], [0, 1], n)
```

Exercise: Both the above definitions of FIB unnecessarily compute one more Fibonacci number than desired. Modify the latter definition of FIB to avoid this extra calculation. However, note that the modified definition will *not* give a result in agreement with the recurrence definition for $n=0$.

> **Hint**: Change the last argument of ITERATE to $n-1$ and use SUB to extract the *second* element of the result.

Exercise: Chebychev polynomials of the first kind are defined by the recurrence

$$\begin{aligned} T(n, x) &= 1 && \text{if } n = 0, \\ T(n, x) &= x && \text{if } n = 1, \text{ and} \\ T(n, x) &= 2 \cdot x \cdot T(n-1, x) - T(n-2, x) && \text{if } n > 1. \end{aligned}$$

Define a function for $T(n, x)$ using ITERATE and test it for speed and correctness against the definition of CHEBYCHEV_T in the file ORTH_POL.MTH (see Section 9.17).

For additional examples of iteration, inspect the definitions of the functions

- FIXED_POINT and NEWTONS in the file SOLVE.MTH,
- IMP_DIF and PARA_DIF in the file DIF_APPS.MTH, and
- PICARD, EULER, TAYLOR_ODE1, TAYLOR_ODES and RK in the file ODE_APPR.MTH.

10.3 IF Expressions

Often functions are defined in a piecewise manner using one or more conditions. For example, a job may pay $10 per hour for the first 40 hours per week and $15 per hour for any additional hours.

IF expressions are a convenient way to express such conditions in ***DERIVE***. For example, if h hours are worked during a week, the above pay schedule can be entered as

```
IF(h ≤ 40, 10h, 400 + 15·(h - 40))
```

When this IF expression is simplified and if ***DERIVE*** determines that the condition $h \le 40$ is true using the current value or domain declaration of h, the second argument $10 \cdot h$ is simplified and returned. If ***DERIVE*** determines that the condition is false, the third argument $400 + 15 \cdot (h - 40)$ is simplified and returned. For example, if h is assigned a value of 50, this IF expression simplifies to 550. Or if h is not assigned a value but is declared less than 30, it simplifies to $10 \cdot h$.

If ***DERIVE*** cannot determine whether the condition is true or false, the entire IF expression is returned.

Note that an IF expression can be used as a subexpression in another expression. For example, adding a $100 bonus to the above pay schedule can be entered as

```
100 + IF(h ≤ 40, 10h, 400 + 15·(h - 40))
```

Exercise: Enter the above IF expression and simplify it. Then declare that h exceeds 40 and resimplify the expression. Finally assign h the value SIN(x) and resimplify the IF expression.

A pair of expressions separated by one of the relational operators (=, /=, <, ≤, >, or ≥) is a relation (see Section 4.13). When a relation is simplified, its two sides are simplified independently and ***DERIVE*** makes no attempt to determine the truth or falsehood of the relation. However, when a relation appears as the first clause of an IF expression and the IF expression is simplified, ***DERIVE*** treats the relation as a condition and actively tries to determine its truth value.

The general form of the IF expression is

IF (*test*, *then*, *else*, *unknown*)

where the *test* clause is a condition and the *then*, *else*, and *unknown* clauses are expressions. If the *test* clause is not a relation or a logical combination of relations (see Section 10.5), it is treated as the condition *test*=0.

When an IF expression is simplified, ***DERIVE*** evaluates the *test* clause to determine its truth value. If true, the *then* clause is simplified and returned as the value of the IF expression. If false, the *else* clause is simplified and returned. If the truth value cannot be determined, the *unknown* clause is simplified and returned. Note that whichever clause is returned, the other two clauses are *not* simplified.

If the *unknown* clause is omitted and the truth value of the *test* clause cannot be determined, the entire IF expression is returned. This is often informative, and subsequent assignments or domain declarations might enable you to resimplify the IF expression to a decisive result. Alternatively, ? is often used as the *unknown* clause since it yields a more compact result when the truth value of the *test* clause cannot be determined. ? is the expression signifying a quantity of unknown phase and magnitude (see Section 4.9).

If the *else* clause is also omitted from the IF expression, a default value of ? is assumed for it. Finally, if the *then* clause is also omitted, IF returns 1 if the argument evaluates to true, otherwise it returns 0. When used in conjunction with the SUM function, this is useful for counting the elements of a vector or sequence that satisfy some condition. For example,

```
SUM(IF(PRIME(n)), n, 1, 100)
```

simplifies to 25, the number of primes between 1 and 100.

Exercise: Clear h of any assigned value or domain declaration, replace with ? the *unknown* clause in the IF expression of the previous exercise, and then simplify the modified IF expression. Next, delete the *else* and *unknown* clauses, and then simplify the modified IF expression. Finally, assign a value to h that makes the *test* clause false, and then simplify again.

Exercise: The acceleration of gravity at a distance $d>0$ from the center of a planet is equal to the acceleration at the surface of the planet multiplied by a factor involving d. Let d be measured in units of the planet's radius (i.e. at the surface $d=1$). Enter an IF expression that returns this factor and plot it in a 2D-plot window for d varying between approximately 0 and 3.

> **Hint**: Above the surface of the planet (i.e. when $d>1$), the factor is $1/d^2$. Below the surface (i.e. when $d<1$), it is d.

An expression of the form

ITERATE (IF (*test*, x, u), x, x_0)

can be used to search for a number or expression that satisfies the search condition *test*. Given x, u generates the next number or expression to test. For example,

```
ITERATE(IF(PRIME(n) AND PRIME(n+2), n, n+2), n, 100001)
```

simplifies to 100151, the leading element of the first prime pair larger than 100000. (Prime pairs are consecutive primes that differ by 2.)

Exercise: Use NEXT_PRIME (see Section 6.13) to improve the performance of the above expression for finding prime pairs. It should reduce the number of integers tested and eliminate one of the two primality tests in the search condition.

IF expressions can be used on the right side of a function definition. For example, most ***DERIVE*** functions are defined for complex as well as real values of their arguments, and many functions return complex results for some real values of their arguments. However, in an educational setting devoted to the real domain, some instructors may prefer that functions return ? when an argument or a result is not real. Using the following filter function, more restrictive variants can be defined that impose this constraint:

```
REAL_ONLY(x) := IF(IM(x) = 0, x)
```

Note that the *else* clause of the IF expression is omitted so that ? is automatically returned if ***DERIVE*** can determine that IM(x) is not 0.

As examples of using REAL_ONLY, you can define:

```
REAL_LN(x) := REAL_ONLY(LN(x))
REAL_ABS(x) := ABS(REAL_ONLY(x))
```

LN(x) is not real-valued if x is negative. ABS(x) is real-valued even if x is not real. These two facts account for the different placement of REAL_ONLY in the above definitions.

Exercise: Define and test REAL_SQRT(x) that returns $\sqrt{x}$ when $\sqrt{x}$ is real, otherwise it returns ? .

Since zero can be approached from either the positive or negative direction, or for that matter any direction in the complex plane, zero does not have a definite sign. Thus, SIGN(0) simplifies to ± 1 (see Section 6.8).

Exercise: Define and test your own sign function named SGN such that SGN(x) simplifies to 1 when x is positive, to -1 when x is negative, and to 0 when x is zero.

For additional examples of IF expressions, inspect the definitions in the files VECTOR.MTH and NUMERIC.MTH.

10.4 Nested IF Expressions

Sometimes multiple conditions are required to define a function. For example, a job may pay \$10 per hour for the first 40 hours per week, \$15 per hour for additional hours through 50 per week, and \$20 per hour for hours exceeding 50 per week.

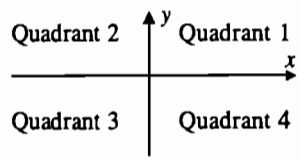
Nested IF expressions are a convenient way to express multiple conditions. Any clause of an IF expression can itself be an IF expression. For example, the above pay schedule can be entered as

```
IF(h ≤ 40, 10h, IF(h ≤ 50, 400 + 15(h − 40), 550 + 20(h − 50)))
```

Exercise: Enter the above expression, and then enter an equivalent one in which the nesting occurs instead in the *then* clause of the outermost IF. Compare the results for *h* assigned values 39, 40, 41, 49, 50, and 51. Why were these test values suggested?

Exercise: A negative *h* in this example is probably a mistake. Therefore, enclose the expression in another IF expression that returns ? when *h* is negative. Note that both the *else* and *unknown* clauses can be omitted from the outermost IF expression.

The axes divide the coordinate plane into four regions called quadrants. The quadrants are numbered as shown:



QUADRANT, as defined below using nested IF expressions, returns the number of the quadrant containing the coordinate point (x, y).

```
QUADRANT(x, y) := IF(x ≥ 0, IF(y ≥ 0, 1, 4), IF(y ≥ 0, 2, 3))
```

Exercise: Enter the above definition of QUADRANT and test it for the points (1,1), (−1,1), (−1,−1), (1,−1), (1,0), (0,1), (−1,0), (0,−1), and (0,0). Why were these particular points suggested?

Note that the above definition of QUADRANT includes the origin and both positive axes in the first quadrant, the negative *x*-axis in the second quadrant, and the negative *y*-axis in the fourth quadrant.

Exercise: Modify QUADRANT so it returns an appropriate phrase such as "Origin" or "Positive x-axis" when the point is on an axis.

IF expressions can be nested arbitrarily deep, however, it is usually better to subdivide complicated IF expressions into more comprehensible components by defining auxiliary helper functions.

10.5 The Logical Operators

Sometimes the value of a function depends on both of two conditions being true. For example, a job may pay a flat $10 per hour for any number of hours per week. Obviously, at least 0 hours will be worked *and* at most 168 hours can be worked (i.e. the number of hours in a week). Any number of hours outside that range must be a mistake.

AND is the logical operator that makes it possible to express such conjunctions of conditions. For example, if *h* hours are worked during a week, the following IF expression returns the correct pay when *h* is a valid number of hours, and it returns ? when *h* is not valid:

```
IF(0 ≤ h AND h ≤ 168, 10h, ?)
```

When an IF expression is simplified and AND is the top-level operator of the *test* clause, the truth value of the *test* clause is determined as follows: The left operand of the AND is evaluated and

- If its truth value is false, the value of the *test* clause is also false. Note that in this case the right operand of the AND is *not* evaluated.
- If its truth value is true, the right operand of the AND is evaluated and returned as the value of the *test* clause.
- If its truth value cannot be determined, the right operand of the AND is evaluated and if it is false, the value of the *test* clause is also false. Otherwise, the truth value of the *test* clause cannot be determined.

Exercise: A triangle is equilateral if its three angles are equal. Define and test a function that returns "Equilateral" if its three arguments are equal, and "Not equilateral" if the three arguments are not all equal.

Sometimes the value of a function depends on either of two conditions being true. OR is the logical operator that makes it possible to express such disjunctions of conditions. For example, a triangle is isosceles if any two of its angles are equal. CATEGORY is a function that returns "Isosceles" or "Scalene" depending on the three angles *a*, *b* and *c*:

```
CATEGORY(a,b,c) := IF(a=b OR a=c OR b=c, "Isosceles", "Scalene")
```

Exercise: Enter the above definition, then simplify CATEGORY(1,1,1).

The result of this exercise indicates that, in ***DERIVE***, the OR operator corresponds to the phrase "and/or" in English. Thus, CATEGORY returns "Isosceles" even if all three angles are equal.

Exercise: Using nested IF expressions and logical operators, redefine CATEGORY so it returns "Equilateral", "Isosceles", or "Scalene". You can make use of the equilateral test function of the preceding section if you saved its definition.

When an IF expression is simplified and OR is the top-level operator of the *test* clause, the truth value of the *test* clause is determined as follows: The left operand of the OR is evaluated and

- If its truth value is true, the value of the *test* clause is also true. Note that in this case the right operand of the OR is *not* evaluated.
- If its truth value is false, the right operand of the OR is evaluated and returned as the value of the *test* clause.
- If its truth value cannot be determined, the right operand of the OR is evaluated and if it is true, the value of the *test* clause is also true. Otherwise, the truth value of the *test* clause cannot be determined.

Exercise: A triangle is a right triangle if any of its three angles is $\pi/2$. Define and test a function that returns "Right" if one of its angles is $\pi/2$, and "Not right" if none of its angles is $\pi/2$.

Each operand of AND or OR can itself be an expression involving AND or OR. For example, three values a, b and c are a monotonic sequence if $a \le b \le c$ or $a \ge b \ge c$. The following IF expression tests for monotonicity:

```
IF(a≤b AND b≤c OR a≥b AND b≥c, "Monotonic", "Nonmonotonic")
```

In many ways AND is analogous to the * operator, and OR is analogous to the + operator. For this reason AND has a higher precedence than OR, so the above *test* clause is interpreted as if it were parenthesized

```
(a ≤ b AND b ≤ c) OR (a ≥ b AND b ≥ c)
```

rather than

```
a ≤ b AND (b ≤ c OR a ≥ b) AND b ≥ c
```

Exercise: Three values a, b, and c are a **strict monotonic** sequence if $a<b<c$ or $a>b>c$. Define and test a function that returns "Monotonic", "Strict monotonic", or "Nonmonotonic" if its three arguments are a monotonic, a strict monotonic, or a nonmonotonic sequence.

Boolean expressions are those expressions involving logical operators. Boolean expressions have many equivalent forms. To avoid errors and save time and space, it is worth trying to find the simplest possible Boolean expression for use as the *test* clause of IF expressions.

The following identities, valid for any *u*, *v*, and *w*, are useful for simplifying Boolean expressions:

```
u AND v   ↔  v AND u
u OR v    ↔  v OR u

u AND (v AND w)   ↔  (u AND v) AND w
u OR (v OR w)     ↔  (u OR v) OR w

u AND (v OR w)    ↔  u AND v OR u AND w
u OR v AND w      ↔  (u OR v) AND (u OR w)
```

The first two identities indicate that AND and OR are commutative. The second two indicate that AND and OR are associative. The last two indicate that AND distributes over OR and that OR distributes over AND.

For simplifying Boolean expressions, these identities are most often useful in the right-to-left direction. For example, the Boolean expression

```
(x>0 AND y=3) OR (x>0 AND z=5)
```

can be factored into the more concise expression

```
x>0 AND (y=3 OR z=5)
```

Similarly

```
(x>0 OR y=3) AND (x>0 OR z=5)
```

can be factored into the more concise expression

```
x>0 OR (y=3 AND z=5)
```

Exercise: Manually reduce the expression

```
(x=0 AND y=1) OR (y=2 AND x=0) OR (x=0 AND y=3)
```

to a simpler form.

The NOT logical operator negates the truth value of its operand: NOT is true if its operand is false, and false if its operand is true. For example, NOT($x > 5$) is true if x is not greater than 5.

Since each relational operator has a complementary operator, the negation of a *test* clause can be transformed into a more readable and efficient one using the complementary operator. For example, NOT($x > 5$) is equivalent to $x \leq 5$, and NOT($x=5$) is equivalent to $x/=5$.

In many ways NOT is analogous to the – minus operator, so it is not surprising that for any Boolean expression *u*,

```
NOT NOT u  ↔  u
```

De Morgan's laws allow NOT to be distributed over both AND and OR. For any expressions *u* and *v*:

```
NOT(u AND v)  ↔  (NOT u) OR (NOT v)
NOT(u OR v)  ↔  (NOT u) AND (NOT v)
```

By repeatedly applying the above rules and using complementary relation operators, the NOT operator can always be eliminated from Boolean expressions.

Exercise: Manually reduce the Boolean expression

```
NOT(a > 5 OR b = 7 AND c /= 9)
```

to a simpler form.

10.6 Recursive Functions

A function definition is recursive if the right side of the definition calls the function being defined. For example, if n is a nonnegative integer, the factorial of n can be computed recursively by the rules

$$n! = 1 \qquad \text{if } n = 0, \text{ and}$$
$$n! = n \cdot (n-1)! \qquad \text{if } n > 0.$$

The first rule is the terminal case for this definition. The second rule is the recursive case. The ***DERIVE*** definition corresponding to the above rules is

```
FACT(n) := IF(n = 0, 1, n·FACT(n - 1))
```

To be computable, recursive definitions must have one or more terminal case rules, and the recursive case rules must make finite progress toward the terminal cases. The above definition for computing factorials is computable because

- When the terminal case rule is applicable (i.e. when $n=0$), the recursion is terminated.
- When the recursive case rule is applicable (i.e. when $n>0$), the problem is reduced to one closer to the terminal case.
- The terminal case is reached after a finite number of applications of the recursive case rule.

Exercise: Enter the above definition of FACT. Compare its speed with the pre-defined factorial operator ! on test cases such as n = 0, 1, 2, 64, 128, 256, -1, 1/2, 3/2, and on an unassigned variable m.

If x is an expression and n is a nonnegative integer, raising x to the n power can be computed recursively by the rules

$$\text{RAISE}(x, n) = 1 \qquad \text{if } n = 0, \text{ and}$$
$$\text{RAISE}(x, n) = x \cdot \text{RAISE}(x, n-1) \qquad \text{if } n > 0.$$

Exercise: Define and test a recursive function RAISE corresponding to the above rules. Compare its speed with the iterative function POWER defined in Section 10.2 by using both functions to compute b^0, b^1, b^2, b^{64}, b^{128} and b^{256} where $b=1234567890$.

Exercise: Extend the definition of RAISE to handle negative integer powers by using reciprocals, and test the modified RAISE by computing 2^{-1}, 2^{-2} and 2^{-3}.

If m and n are integers, the greatest common divisor (gcd) of m and n can be computed recursively by the rules

$$\begin{aligned} \text{GCD}(m, n) &= |m| && \text{if } n = 0, \text{ and} \\ \text{GCD}(m, n) &= \text{GCD}(n, |m-n|) && \text{if } n \neq 0. \end{aligned}$$

The recursion terminates because both arguments must become nonnegative after two recursions, and thereafter the second argument decreases toward the terminal case by at least 1 for each recursion. It is also true that any integer that evenly divides m and n, also evenly divides n and $|m - n|$, including the greatest common divisor. It is also true that the greatest common divisor of m and 0 is $|m|$.

Exercise: Define and test a recursive function G_C_D corresponding to the above rules. Note that you cannot name the function GCD since that is a pre-defined function (see Section 6.7).

The tests carried out in the last several exercises should indicate that the speeds of FACT and ! are approximately proportional for large n, as are the speeds of RAISE, POWER, and ^. Although pre-defined operators or functions are faster, recursively defined functions can be reasonably efficient as well. The next example illustrates the importance of writing efficient definitions.

If n is a nonnegative integer, the nth Fibonacci number can be computed recursively by the rules

$$\begin{aligned} \text{FIB}(n) &= 0 && \text{if } n = 0, \\ \text{FIB}(n) &= 1 && \text{if } n = 1, \text{ and} \\ \text{FIB}(n) &= \text{FIB}(n-1) + \text{FIB}(n-2) && \text{if } n > 1. \end{aligned}$$

An iterative definition of FIB that implemented these rules using ITERATE was discussed in Section 10.2. The following recursive definition of FIB_SLOW is another way to implement this recurrence:

```
FIB_SLOW(n) := IF(n < 2, n, FIB_SLOW(n - 1) + FIB_SLOW(n - 2))
```

Exercise: Enter the above definition of FIB_SLOW. Compare the time required to compute FIB_SLOW(n) with that of the iteratively defined FIB(n) for n = 2, 4, 8, 16, etc. until you are convinced that the FIB_SLOW times grows much more dramatically with n.

The reason FIB_SLOW is so slow is because to compute FIB_SLOW(n), FIB_SLOW(n–2) is computed twice: once for FIB_SLOW(n) and once for FIB_SLOW(n–1). Similarly, FIB_SLOW(n–3) is computed three times: once for FIB_SLOW(n–1) and once for each of the two computations of FIB_SLOW(n–2). In general, FIB_SLOW(n–i) must be computed the ith Fibonacci number of times. Thus

the number of redundant calculations, and hence the time required to compute the *n*th Fibonacci number using FIB_SLOW increases rapidly (i.e. Fibonaccially!) with *n*.

Such redundant computation is typical for recursive definitions that naively implement recurrences, like Fibonacci, that are doubly recursive. Note that the recurrences for factorial, raising to a power, and gcd were just singly recursive and so did not suffer such redundancy.

In the case of Fibonacci this inefficiency is easy to overcome: Just define an auxiliary helper function that has extra accumulation arguments to retain previously computed values so that they do not have to be recomputed. The main function then invokes this helper function with the appropriate starting arguments.

```
FIB_AUX(n, f1, f0) := IF(n=0, f0, FIB_AUX(n - 1, f1 + f0, f1))

FIB_FAST(n) := FIB_AUX(n, 1, 0)
```

Exercise: Enter the above definitions of FIB_AUX and FIB_FAST. Compare the time required to compute FIB_FAST(*n*) with that of the iteratively defined FIB(*n*) for $n = 2, 4, 8, 16, \ldots$ until you are convinced that their speeds are both acceptable and approximately proportional.

Exercise: The above definition of FIB_FAST unnecessarily computes one more Fibonacci number than desired. This can be avoided by terminating with the value of *f1* rather than the value of *f0*, provided FIB_FAST invokes FIB_AUX with appropriate values. Modify the definitions accordingly.

Exercise: The last exercise in Section 10.2 defines Chebychev polynomials of the first kind by a two-term recurrence. Define and test a direct recursive function for generating these polynomials. Then define and test an alternative recursive version that uses a helper function with accumulation variables. Compare the speed of two recursive functions with the iterative function you defined in Section 10.2.

For additional examples of recursive functions, see the definition of POLY_DEGREE in the file MISC.MTH or the definitions of SPHERICAL_BESSEL_J and SPHERICAL_BESSEL_Y in the file BESSEL.MTH.

10.7 Mutual Recursion

Sometimes a function invokes itself recursively via another one or more intermediate functions rather than directly. This is all right provided at least one of the mutually recursive functions has an IF expression that eventually terminates the recursion. The F and G series from celestial mechanics provide an interesting example. This series can be used to predict a conic trajectory of a comet given its position and velocity at one time. The series is defined by

$$F(0) := 1$$

$$G(0) := 0$$

$$F(n) := m \cdot \frac{d}{d\mu} F(n-1) + s \cdot \frac{d}{d\sigma} F(n-1) + e \cdot \frac{d}{d\epsilon} F(n-1) - \mu \cdot G(n-1)$$

$$G(n) := m \cdot \frac{d}{d\mu} G(n-1) + s \cdot \frac{d}{d\sigma} G(n-1) + e \cdot \frac{d}{d\epsilon} G(n-1) + F(n-1)$$

for $n > 0$, where

$$m := -3 \cdot \mu \cdot \sigma$$

$$s := \epsilon - 2 \cdot \sigma^2$$

and

$$e := -\sigma \cdot (\mu + 2 \cdot \epsilon)$$

To define a pair of mutually recursive functions F and G for computing the series, first make G an arbitrary function by the assignment

$$G(n) :=$$

then define F and G as follows:

$$F(n) :=$$

$$\text{IF}\left[n=0,\ 1,\ m \cdot \frac{d}{d\mu} F(n-1) + s \cdot \frac{d}{d\sigma} F(n-1) + e \cdot \frac{d}{d\epsilon} F(n-1) - \mu \cdot G(n-1)\right]$$

and

$$G(n) :=$$

$$\text{IF}\left[n=0,\ 0,\ m \cdot \frac{d}{d\mu} G(n-1) + s \cdot \frac{d}{d\sigma} G(n-1) + e \cdot \frac{d}{d\epsilon} G(n-1) + F(n-1)\right]$$

Making G an arbitrary function is necessary so that the G that occurs in the definition of F is interpreted as a function rather than a variable. G is assigned its actual definition by the final assignment above.

Exercise: Enter the above definitions for F and G and then verify that F(4) simplifies to

$$\mu \cdot (3 \cdot \epsilon + \mu - 15 \cdot \sigma^2)$$

Finally, record how the computing time for F(*n*) grows with *n*.

The above definition causes exponential recomputation of the same formulas: F(*n*) requires F(*n*–1) and G(*n*–1), both of which redundantly compute F(*n*–2) and G(*n*–2), etc. Thus it is worth redefining the recursion to use an auxiliary helper function that carries forward F(*n*–1) and G(*n*–1) in auxiliary variables. The two final values F(*n*) and G(*n*) can be bundled together in a two-element vector and returned as the value of the function. First, clear the definition of F and G and make them variables again by the assignments

```
f :=
g :=
```

Then define

```
FG_AUX(n, f, g) := IF[n=0, [f, g], FG_AUX[n-1,
```

$$m \cdot \frac{d}{d\mu} f + s \cdot \frac{d}{d\sigma} f + e \cdot \frac{d}{d\epsilon} f - \mu \cdot g,\ m \cdot \frac{d}{d\mu} g + s \cdot \frac{d}{d\sigma} g + e \cdot \frac{d}{d\epsilon} g + f]]$$

and

```
FG(n) := FG_AUX(n, 1, 0)
```

Then for example, FG(4) simplifies to the vector

$$[\ \mu \cdot (3 \cdot \epsilon + \mu - 15 \cdot \sigma^2),\ 6 \cdot \mu \cdot \sigma\ ]$$

Exercise: Enter the above function definitions and compare the speed of FG(*n*) with the speeds that you recorded for F(*n*).

As a final matter of style in the preceding example, it would be better to use multiple-character names for the variables *m*, *s* and *e* to reduce the chance of a name conflict with any unassigned variables. For example, use *m_*, *s_* and *e_*.

In this section an example of inefficient mutual recursion was given, and then an efficient direct recursion alternative. Do not conclude from this that all mutual recursion is inefficient. In fact, ***DERIVE*** itself is a large collection of mutually recursive ***muLISP*** functions! The ***muLISP LISP Language Programming Environment***, also created by Soft Warehouse, Inc., is available through dealers.

10.8 Epilogue

Programming is like chess. Learning the rules of the game is easy, but it takes a great deal of practice and diligence to become proficient at it. Mathematical experience should make *DERIVE*'s built-in capabilities relatively easy and straightforward to use. However, if you are not experienced at programming, be prepared to write lots of simple *DERIVE* programs before attempting more ambitious ones. And most important: test your programs thoroughly!

Questions and Answers

This appendix answers some of the questions commonly asked by new users of ***DERIVE***. Also provided is the section number where more detailed information can be found. The questions and answers are divided into five sections: user interface, simplification, 2D plot, 3D plot, and printing.

User Interface Questions

Q: If I decide not to issue a command, how do I abort the command and escape from the submenu?

A: Press the **Esc** key until you return to the top-level command menu (see Section 2.3).

Q: While line editing, how do I insert characters in a line without over-writing the characters that are already there?

A: Press the **Ins** key to switch to insert mode. The abbreviation Ins will appear on the status line on the bottom of the screen (see Section 2.4).

Q: Why does ***DERIVE*** periodically pause in the display of expressions and plots?

A: When ***DERIVE*** runs out of memory, it recycles memory occupied by data structures that are no longer needed. During the time it takes for this recycling, other processing stops. On PC compatible computers, a full-block cursor is displayed on the first character of the message line (see Section 2.2) during this process.

Q: Why doesn't the percentage of free space displayed on the status line change as soon as I generate or remove expressions?

A: This percentage is *not* updated until ***DERIVE*** recycles memory (see Section 2.2).

Q: How do I get a summary of the current system control settings?

A: Use the **Help System** command to display the current setting of all the system control variables (see Sections 2.5 and 2.14).

Q: How do I display a DOS file directory from within ***DERIVE***?

A: Issue the **Options Execute** command and then enter DIR/W at the DOS prompt (see Section 2.7). Also, after issuing a **Transfer Load** command, you can press **F1** to display and choose a file from a DOS directory (see Section 2.9).

Q: Why do the direction keys sometimes stop working?

A: You probably pressed the **NumLock** key. Press it again to toggle it off.

Q: How do I access the elements of a vector or matrix?

A: Use the arrow keys to highlight the desired element, and press the **F3** or **F4** function key to insert the element on the author line (see Section 3.3). A highlighted vector or matrix element can also be used as an operand of the **Build** command (see Section 3.5). The SUB operator or the ELEMENT function can also be used to extract elements (see Section 8.3).

Simplification Questions

Q: Why doesn't LN(e) simplify to 1?

A: In ***DERIVE***, e is a simple variable, *not* the base of the natural logarithms (2.71828...). The base of the natural logarithms is displayed as ê and it is entered by typing #e or by pressing **Alt-E** on PC compatible computers (see Section 6.1).

Q: Why doesn't i^2 simplify to -1?

A: In ***DERIVE***, i is a simple variable, *not* the imaginary unit ($\sqrt{-1}$). The imaginary unit is displayed as î and it is entered by typing #i or by pressing **Alt-I** on PC compatible computers (see Section 6.8).

Q: How do I turn the name of a function I defined back into a variable name?

A: Use the **Declare Variable** command to declare the name a real-valued variable (see Section 4.10).

Q: Why doesn't ***DERIVE*** simplify $\text{LN}(x^2-x) - \text{LN}(x)$ to $\text{LN}(x-1)$?

A: This transformation is invalid if x is negative. However, if you use the **Declare Variable** command to declare that x is nonnegative, ***DERIVE*** will simplify the expression. In general, ***DERIVE*** will *not* use a transformation unless it can determine that the transformation is valid (see Section 4.10).

Q: When I use square brackets when entering expressions like 3 [x+y], why does the result not behave correctly?

A: In ***DERIVE***, square brackets are used *exclusively* for entering vectors and matrices (see Section 8.1). Thus the above expression is interpreted as 3 times the one element vector [x+y]. Parentheses rather than brackets should be used to control the order in which operators are applied (see Section 3.1). Note that ***DERIVE*** has to display multi-line parentheses using square brackets, because large parentheses are not part of the extended ASCII character set.

Q: When I integrate the derivative of $x/(x+1)$, why do I get $-1/(x+1)$ instead of the original expression?

A: Antiderivatives are not unique and may differ by a constant. In this case $x/(x+1)$ and $-1/(x+1)$ differ by 1, and both are valid antiderivatives.

2D Plot Questions

Q: How can I adjust the aspect ratio used for 2D plots?

A: Use the **aXes** command and adjust the Rows and Columns fields (see Section 5.1.2).

Q: For 2D plots, how do I change the region being plotted?

A: Use the **Range** command. Alternatively, you can move the cross near the desired center of the plot, zoom in or out as desired, and then issue a **Center** command (see Section 5.1.3).

Q: How do I make a 2D plot of a function implicitly defined by an equation such as $x^2+y^2=1$?

A: Highlight the equation and plot it in a 2D-plot window just as you would an explicitly defined function (see Section 5.1.9).

Q: Why can't I plot $x^{1/3}$ for negative x?

A: The principal cube root of a negative number is complex and cannot be plotted directly. Use the **Manage Branch** command to make ***DERIVE*** select the real branch instead of the principal branch (see Section 4.5).

Q: When I try to plot a vector of two expressions, why doesn't ***DERIVE*** plot them correctly?

A: ***DERIVE*** plots a vector of two univariate expressions parametrically (see Section 5.1.7). However, you can add ? as a dummy third element to the vector so ***DERIVE*** produces the multiple plot you desire.

Q: My computer has a Hercules compatible graphics card. Why don't I get Hercules graphics when I select it using the **Options Display** command?

A: Run the HERCULES.COM resident program *before* starting ***DERIVE*** (see Section 2.12).

Q: When I switch to graphics display mode, why are strange characters displayed on the screen for integral signs, built-up fractions, Greek letters, and the borders of the window?

A: To display characters in the extended ASCII character set in graphics mode, some computers (e.g. the AT&T 6300 and the Toshiba 3100) require that a graphics table be loaded into memory *before* starting ***DERIVE***. The program to load the table may be called GRAFTABL.COM or GFTABLE.COM (see your computer manual for details). If no such program is available, use the **Options Display** command and select the standard character set (see Section 2.12).

3D Plot Questions

Q: Why is my 3D plot just a coarse collection of dots?

A: For a higher resolution plot, switch from text to a graphics display mode (see Section 2.12).

Q: How can I make a 3D plot of two intersecting surfaces?

A: You can't. However, plotting the minimum of the two surfaces allows you to see their curve of intersection. For example, try plotting MIN(x^2+y^2, 15).

Q: What can I do if a Memory Full warning message appears while plotting in 3D?

A: Reduce the number of grid lines (see Section 5.2.1), close unneeded windows (see Section 2.13), and/or remove expressions from the algebra window (see Section 3.7).

Q: How do I reset all the plot window parameters to their default values?

A: Use the **Window Designate** command to replace the current 3D-plot window with a new one (see Section 2.13).

Printing Questions

Q: How do I print the mathematical expressions in an algebra window?

A: Use the **Transfer Print Expressions Printer** command (see Section 2.10). Naturally your printer must be on line and correctly connected to your computer (see your DOS and printer manuals for details).

Q: Why aren't mathematical expressions printed using integral signs and Greek letters?

A: If your printer is capable of printing the extended ASCII character set, use the **Transfer Print Expressions Options** command and select the Extended character set (see Section 2.10).

Q: Why don't I get a good printout when I use the **Transfer Print Screen Printer** command or press **Shift-F10** to print an image of the ***DERIVE*** screen?

A: Before printing, use the **Transfer Print Options** command to select the proper printer type (Laserjet, Dotmatrix, or color Deskjet), and use the **Options Display** command to select a graphics display mode (see Section 2.12).

Q: How can I incorporate an image of a ***DERIVE*** screen into a manuscript?

A: Most popular word processors can import graphics images in the form of TIF (Tagged Image Format) files (see your word processor manual for details). Use the **Transfer Print Screen File** command or press **Ctrl-F10** to save an image of the current ***DERIVE*** screen as a TIF file (see Section 2.11). To save a screen image, first use the **Options Display** command to switch to graphics mode (see Section 2.12).

Screen Capture Programs

Screen capture programs make it possible to send graphics screen images to your printer. Some also make it possible to merge such screen images into documents produced on some word processors. These programs are useful with any program, including ***DERIVE***, from which you want to capture a screen image.

Most screen capture programs are resident programs that must be run *before* you start ***DERIVE***. On PC compatible computers, they are usually activated by pressing **Shift-PrtSc**.

GRAPHICS.COM is a screen capture program distributed free with MS-DOS. The DOS 2 and DOS 3 version of GRAPHICS.COM is capable of printing CGA graphics screens on some popular dot-matrix printers. The DOS 4 version is capable of printing CGA, EGA, and VGA graphics screens on some popular dot-matrix printers. The DOS 5 and DOS 6 versions are capable of printing CGA, EGA, and VGA graphics screens on many popular dot-matrix *and* laser printers. See your MS-DOS manual for details.

Some word processors such as **Microsoft Word®** version 5.0 or later and **Word Perfect™** version 5.1 or later come with screen capture programs especially suited for incorporating screen images in documents.

The following, in alphabetical order, are some of the commercial screen capture programs capable of printing graphics screens produced by ***DERIVE***:

- **Collage Plus™**: Inner Media Inc., 60 Plain Road, Hollis, NH, 03049, USA, phone (603) 465-3216, fax (603) 465-7195.
- **GRAFPLUS™**: Jewell Technologies, Inc., 4740 44th Ave. S.W., Suite 203, Seattle, WA, 98116, USA, phone (206) 937-1081, fax (206) 937-1507.
- **Hotshot Graphics™**: Symsoft Corp., 444 First Street, Los Altos, CA, 94022, USA, phone (415) 941-1552, fax (415) 941-1417.
- **Pizazz Plus™**: Application Techniques, 10 Lomar Park Drive, Pepperell, MA, 01463, USA, phone (800) 433-5201 or (508) 433-5201, fax (508) 433-8466.
- **Raindrop™**: Eclectic Systems, 8106 St. David Court, Springfield, VA, 22153, USA, phone (703) 440-0064.

While not a screen capture program, the **MathPORT™** utility program makes possible the typeset quality printing of mathematical formulas produced by ***DERIVE***. **MathPORT** converts ***DERIVE*** MTH files, as well as files produced by other popular math programs, into TIF (Tagged Image Format) files and/or TEX files. It is available from **K-Talk Communications**, 1287 King Avenue, Suite 203, Columbus, OH, 43212, USA, phone (800) 551-7175, fax (614) 488-9505, email ktalk@aol.com

Additional Resources

The Soft Warehouse, Inc. Web Page

For the latest information about ***DERIVE*** and other Soft Warehouse, Inc. products, surf the Internet to Hawaii! At the Soft Warehouse web page (http://www.derive.com) you can

- learn the major capabilities of ***DERIVE*** and ***DERIVE XM***.
- check out the new features in ***DERIVE*** Version 3.
- link to other web pages of interest to ***DERIVE*** users.
- download user contributed MTH files and a ***DERIVE*** demonstration program.
- send problem reports directly to Soft Warehouse, Inc.
- get information on the ***muLISP*** and ***muLISP XM*** programming languages.

The *DERIVE* User Group

An independent, international user group has been established as a forum for the exchange of ideas within the growing community of ***DERIVE*** users. Members receive a subscription to **The *DERIVE* Newsletter** published by the group. The newsletter includes articles written by ***DERIVE*** enthusiasts explaining how to make the best use of the system. The editor always welcomes contributions. For more information, write ***DERIVE* User Group**, Josef Böhm, A-3042 Wurmla, D'Lust 1, Austria; phone/fax 43-(0)2275/8207; email telehaus@noet.at.

The International *DERIVE* Journal

The International *DERIVE* Journal, published by Research Information Ltd, disseminates information about research and practice on the use of ***DERIVE*** as a tool for doing and learning mathematics. The journal aims to enhance the use of ***DERIVE*** by reporting on research and significant innovations.

The journal publishes contributions on all aspects of the use of ***DERIVE*** in education and training which contribute to knowledge in this field and are likely to be of interest to readers. These include research papers, didactic issues, and opinions. Three issues are published per year. For more information visit the **The International *DERIVE* Journal** web page at www.tech.plym.ac.uk/DERIVEHM/DERIVE.HTML (the capitalization is important!).

The editor is Professor John Berry at the Centre for Teaching Mathematics, The University of Plymouth, England. His email address is jberry@plymouth.ac.uk. To subscribe to the journal contact Research Information Ltd at 222 Maylands Avenue, Hemel Hempstead, Herts, HP2 7TD, United Kingdom; phone +44(0)1442 213222; fax +44(0)1442 259395.

The *DERIVE* Internet Mailing List

An independently operated electronic mailing list has been established on the Internet to promote the discussion of issues related to ***DERIVE***. To join the mailing list, send a one line e-mail message of the form

JOIN derive-news *YourFirstName YourLastName*

to the address mailbase@mailbase.ac.uk. To contribute to the derive-news mailing list, send e-mail to the address derive-news@mailbase.ac.uk. For more information about derive-news, contact the list owner through e-mail at derive-news-request@mailbase.ac.uk.

The *DERIVE* Electronic Bulletin Board

An independently operated electronic bulletin board has been established for the exchange of information concerning ***DERIVE***. The bulletin board phone number in Illinois, USA is (217) 337-0926. The board supports 1200, 2400, 9600, and 14400 baud modems (no parity, 8 data bits, 1 stop bit). For more information, call MathWare at (800) 255-2468 or fax them at (217) 384-7043.

English Language Books Using *DERIVE*

- Arney, David C.: **Exploring Calculus with *DERIVE***, Addison-Wesley Publishing Company, Route 128, Reading, MA, 01867, USA, 1992, ISBN 0-201-52839-8, phone (617) 944-3700. This book shows how to use ***DERIVE*** to solve calculus and related problems. It is a companion to any of the textbooks used in calculus courses. The author hopes that it "can contribute to better student understanding and performance in problem solving." 166 pages.

- Arney, David C.: **Differential Equations with *DERIVE***, 1993, MathWare, P.O. Box 3025, Urbana, IL, 61801, USA, phone (800) 255-2468, fax (217) 384-7043, ISBN 0-9623629-3-X, Paperbound. This book of examples and exercises organized by topic shows how to solve and analyze differential equations using ***DERIVE***. It is meant as a companion manual for any differential equation textbook. Both elementary and intermediate methods and techniques are discussed for both ordinary and partial differential equations. 284 pages.

- Berry, John S.; Graham, Edward; Watkins, Anthony J.P.: **Learning Mathematics Through *DERIVE***, Ellis Horwood Limited, Market Cross House, Cooper Street, Chichester, West Sussex, PO19 1EB, Great Britain, 1993, ISBN 0-13-037532-2. This book develops the foundation of mathematics needed by scientists and engineers through the use of ***DERIVE***. The authors emphasize the role of ***DERIVE*** as an investigative tool to help students understand basic mathematical concepts and as a problem solving tool for solving real-world problems. 371 pages.

- Böhm, Josef, editor: **Teaching Mathematics with *DERIVE***, Proceedings of the International School on the Didactics of Computer Algebra (April 27-30, 1992, Krems, Austria), Chartwell-Bratt Ltd, Old Orchard, Bickley Road, Bromley, Kent BR1 2NE, England, Great Britain, 1992, ISBN 0-86238-319-6. "This illustrated guide is packed full of sound advice and teaching materials. Top European educationalists show how to use computer algebra systems to teach mathematics (particularly to 12-18 year olds)." 300 pages.

- DeMarois, Phil: **College Algebra Laboratories Using *DERIVE***, 1992, MathWare, P.O. Box 3025, Urbana, IL, 61801, USA, phone (800) 255-2468, fax (217) 384-7043. This book lets college algebra students "participate in a series of guided experimental activities promoting the discovery of important mathematical ideas." Topics include equation and inequality solving, linear and quadratic functions, exponential and logarithmic functions, sequences and series, matrices, and the binomial theorem. 125 pages.

- Denton, Brian H.: **Learning Linear Algebra Through *DERIVE***, Prentice Hall International (UK) Limited, Campus 400, Maylands Avenue, Hemel Hempstead, Hertfordshire HP2 7EZ, Great Britain, 1995, ISBN 0-13-122664-9. This book

uses ***DERIVE*** to reinforce learning the theory of linear algebra and make applications more realistic. 353 pages.

- Ellis, Wade Jr. and Lodi, Ed: **A Tutorial Introduction to *DERIVE***, Brooks/Cole Publishing Co., 511 Forest Lodge Road, Pacific Grove, CA, 93950, USA, 1991, ISBN 0-534-15522-7, phone (408) 373-0728, fax (408) 375-6414. This friendly introduction to ***DERIVE*** includes examples and exercises covering equation solving, function definitions, plotting, calculus, and linear algebra. 94 pages.

- Evans, Benny and Johnson, Jerry: **Exploring Calculus with *DERIVE***, John Wiley and Sons, Inc., 605 Third Avenue, New York, NY, 10158, USA, 1994, ISBN 0-471-31048-4. This manual is an enrichment supplement to the Calculus text by Hughes-Hallet, Gleason *et al.*, which was the result of a major NSF initiative in Calculus reform called The Calculus Consortium based at Harvard. 314 pages.

- Evans, Benny and Johnson, Jerry: **Linear Algebra with *DERIVE***, John Wiley and Sons, Inc., 605 Third Avenue, New York, NY, 10158, USA, 1994, ISBN 0-471-59194-7. Available through CipherSystems, 3468 San Juan Circle, Reno, NV, 89509, USA, phone (702) 329-4424, fax (702) 329-4427. This book is a supplement to a traditional linear algebra course. It enables students (and teachers) to use ***DERIVE*** as a tool to to solve problems beyond the level of rote calculations and template exercises. 280 pages.

- Gilligan, Lawrence G. and Marquardt, James F. Sr: **Calculus and the *DERIVE* Program: Experiments with the Computer, Third Edition**, Gilmar Publishing Company, 1995, ISBN 0-9626661-8-1. Available from MathWare, P.O. Box 3025, Urbana, IL, 61801, USA, phone (800) 255-2468, fax (217) 384-7043. This book consists of a brief introduction to ***DERIVE*** followed by 23 calculus experiments using the software. It provides enough material for two to three semesters of calculus. Each experiment includes a list of objectives, background information, a step-by-step procedure, and a data sheet with questions and space for answers. 170 pages.

- Glynn, Jerry: **Exploring Math from Algebra to Calculus with *DERIVE*, A Mathematical Assistant**, 1989, MathWare, P.O. Box 3025, Urbana, IL, 61801, USA, phone (800) 255-2468, fax (217) 384-7043, ISBN 0-9623629-0-5. Using illustrated examples and keystroke guides, this entertaining and enlightening book shows you exactly what to expect when using ***DERIVE***. The author states a mathematical question, walks you through its solution, and then encourages you to extend the problem and charge off in your own direction. Subjects include plotting, factoring, equation solving, complex numbers, trigonometry, calculus, and how to teach math using ***DERIVE***. 144 pages.

- Harper, David; Wooff, Chris; and Hodgkinson, David: **A Guide To Computer Algebra Systems**, John Wiley & Sons Ltd., Baffins Lane, Chichester, West Sussex, PO19 1UD, England, Great Britain, 1991, ISBN 0-471-92910-7. This book is intended as "a thorough and independent comparison of the five major (commercially available computer algebra) systems". It includes numerous tables that show the ability of each of the five systems to answer specific math problems. 148 pages.

- Hill, R.J.; and Keagy T: **Elementary Linear Algebra with *DERIVE*: an integrated text**, Chartwell-Bratt Ltd, Old Orchard, Bickley Road, Bromley, Kent, BR1 2NE, England, Great Britain, 1995, ISBN 0-86238-403-6, phone (+44) (0)181 467 1956, fax (+44) (0)181 467 1754. Using ***DERIVE*** to eliminate time consuming calculations, this book covers linear algebra from determinants to eigenvectors and matrix analysis. The text includes 180 worked-out examples, 600 ***DERIVE*** statements, and 600 exercises with solutions to most. An accompanying diskette provides all the ***DERIVE*** procedures used in the book. 380 pages.

- Hodes, Elizabeth; Mallen, Michael; and Yuhn, M. Paige: **Exploring Precalculus with *DERIVE***, West Publishing Company, 610 Opperman Drive, P.O. Box 64526, St. Paul, MN, 55164-0526, USA, 1994, ISBN 0-314-03892-2. This book consists of 27 explorations using ***DERIVE***. It can be used with a wide variety of college math courses including algebra, trigonometry, and precalculus. 300 pages.

- Heugl, Helmut and Kutzler, Bernhard; editors: ***DERIVE* in Education**, Proceedings of the Second International Conference on the Didactics of Computer Algebra (September, 1993, Krems, Austria), Chartwell-Bratt Ltd, Old Orchard, Bickley Road, Bromley, Kent BR1 2NE, England, Great Britain, 1994, ISBN 0-86238-351-X. Educationalists from 14 countries show how various topics can be taught more effectively and how others might no longer be necessary. They explore new ways of making the best use of ***DERIVE*** at schools, colleges, and universities. 302 pages.

- Jaisingh, Lloyd R.: **College Algebra and the *DERIVE* Program**, West Publishing Company, 610 Opperman Drive, P.O. Box 64526, St. Paul, MN, 55164-0526, USA, 1994, ISBN 0-314-03687-3. This manual contains 16 laboratory experiments using ***DERIVE*** that cover a wide range of topics found in any college algebra course. 161 pages.

- Johnson, Jerry and Evans, Benny: **Discovering Calculus with *DERIVE***, Second Edition, John Wiley and Sons, Inc., 605 Third Avenue, New York, NY, 10158, USA, 1995, ISBN 0-471-00972-5. Available through CipherSystems, 3468 San Juan Circle, Reno, NV, 89509, USA, phone (702) 329-4424, fax (702) 329-4427. The books purpose is to help students use the ***DERIVE*** program as a tool to

explore calculus beyond the level of rote calculations and template exercises by providing problems that are different from those one would normally expect to do with nothing but pencil and paper. It spans the calculus sequence from precalculus through multi-variable calculus and elementary differential equations. 314 pages.

- Kirkwood, James R.: **Calculus Projects for *DERIVE***, Wm. C. Brown Publishers, 2460 Kerper Boulevard, Dubuque, IA, 52001, USA, 1994, ISBN 0-697-22888-6. This book consists of 20 ***DERIVE*** projects dealing with calculus of a single variable including one on chaos and one on probability. 190 pages.

- Kulich, Lisa T. and Victor, Barbara: **The *DERIVE* Calculus Workbook**, HarperCollins College Publishers, 10 East 53rd Street, New York, NY, 10022, USA, 1994, ISBN 0-673-99455-4. This book consists of a detailed introduction to ***DERIVE*** followed by 68 assignments for a Calculus lab. The assignments feature realistic scenarios and data, and are designed with the lab class in mind. 240 pages.

- Kutzler, Bernhard: **Mathematics on the PC - Introduction to *DERIVE***, Soft Warehouse Europe, Softwarepark, A-4232 Hagenberg, Austria, 1994, ISBN 3-9500364-1-5. Available from MathWare, P.O. Box 3025, Urbana, IL, 61801, USA, phone (800) 255-2468, fax (217) 384-7043. This book is for learning how to use ***DERIVE*** by private study. It includes hundreds of screen dumps so you will not get lost on your "journey" through several mathematical topics. The examples also provide ideas for using ***DERIVE*** to teach mathematics. 164 pages.

- Leinbach, L. Carl: **Calculus Laboratories Using *DERIVE***, Wadsworth Publishing Company, New in Mathematics Series, 10 Davis Dr., Belmont, CA, 94002, USA, 1991, phone (415) 595-2350, ISBN 0-534-15480-8. This book consists of 21 ***DERIVE*** laboratory exercises for use in a first year calculus course. Each lab is designed so that the student may concentrate on the process or concept, while ***DERIVE*** does the tedious algebraic work. The goal is "to *lead* students to some deep and interesting mathematics and applications of mathematics". 147 pages.

- Mathews, David M.: **Precalculus Investigations Using *DERIVE***, HarperCollins College Publishers, 10 East 53rd Street, New York, NY, 10022, USA, 1994, ISBN 0-673-99097-4. In the same spirit as the national calculus reform movement, this book is intended to supplement any standard text to provide 12 carefully structured interactive learning environments. The laboratories provide students with conceptual understanding of the core topics that constitute precalculus courses.

- Monaghan, John and Etchells, Terence, editors: **Computer Algebra Systems in the Classroom**, Centre for Studies in Science and Mathematical Education,

University of Leeds, Leeds, LS2 9JT, Great Britain, 1993, phone 0532 334611, ISBN 0 904421 52 X. Based on studies of ***DERIVE*** in the classroom, this collection of six articles is "a timely collection of insights into the transformations that may occur because of the advances in technology and is to be warmly welcomed. It augurs curriculum change at least as fundamental as those from logarithm tables and slide rules to scientific calculators." 82 pages.

- Olwell, David H. and Driscoll, Patrick J.: **Calculus & *DERIVE***, Saunders College Publishing, Harcourt Brace Jovanovich, Publishers, 8th Floor, Orlando, FL, 32887, USA, 1992, ISBN 0-03-076156-5. This book consists of 32 exercises and 4 projects covering the typical three semester calculus sequence through 2nd order differential equations. The exercises require about 45 minutes and the projects about 4 hours to complete. 154 pages.

- Salter, Mary and Gilligan, Lawrence: **Linear Algebra Experiments Using the *DERIVE* Program**, Gilmar Publishing Company, P. O. Box 6376, Cincinnati, OH, 45206, USA, 1992. Each experiment consists of a list of objectives, background information, step-by-step instructions, and a data sheet for questions and answers. Students were given about a week to complete each experiment. 122 pages.

- Schonefeld, Steven A: **Numerical Analysis via *DERIVE***, 1994, MathWare, P.O. Box 3025, Urbana, IL, 61801, USA, phone (800) 255-2468, fax (217) 384-7043, ISBN 0-9623629-2-1. This book shows how to use ***DERIVE*** to implement numerical methods for solving a wide range of problems. It can serve as the primary text for an introductory course in numerical analysis. Topics include: bisection, Newton-Raphson, and secant methods, polynomial interpolation, numerical methods for derivatives, integrals, differential equations, curve fitting, etc. A diskette is included with a number of functions defined in MTH files. 530 pages.

- Watkins, Anthony J.P.: ***DERIVE*-based Investigations for Post-16 Core Mathematics**, Chartwell-Bratt Ltd, Old Orchard, Bickley Road, Bromley, Kent BR1 2NE, England, Great Britain, 1993, ISBN 0-86238-312-9. Tony Watkins of the Centre for Teaching Mathematics, University of Plymouth, England, has produced practical exercises that show how to use ***DERIVE***, and more importantly motivate the students to use ***DERIVE*** as a tool to discover mathematical concepts. 64 pages.

nonEnglish Language Books Using *DERIVE*

- Asai, Osamu: ***DERIVE* Enshu Note, Keisan to gurafu sosa no tame no, yasashii suugaku kyoshitsu**, SEG Corporation, Tokiya Building, 7-13-12 Nishi-Shinjuku, Shinjuku-ku, Tokyo 160, Japan, 1994, ISBN 4-87243-021-2. This exercise workbook is packed full of worked-out mathematical problems using the symbolic and plotting capabilities of ***DERIVE***. Japanese language. 478 pages.

- Asai, Osamu: ***DERIVE* Nyumon**, SEG Corporation, Tokiya Building, 7-13-12 Nishi-Shinjuku, Shinjuku-ku, Tokyo 160, Japan, 1990, ISBN 4-87243-001-8. This book provides an introduction to ***DERIVE***. Japanese language. 68 pages.

- Chadid, Ivan Castro: **Como Hacer Matematicas con *DERIVE***, Editorial Reverte Colombiana, S.A., Calle 37 No. 22-72, Bogota, Colombia, 1992, ISBN 958-95511-0-6. This book discusses algebra, trigonometry, calculus, and differential equations using ***DERIVE*** examples. Spanish language. 446 pages.

- Drijvers, Paul: **Wiskunde leren met *DERIVE* (Learning Mathematics with *DERIVE*)**, Wolters-Noordhoff, The Netherlands, 1992, ISBN 90 01 25992 8. This book has 15 calculus and linear algebra lessons. Dutch language. 80 pages.

- García, Alfonsa, editor: **Prácticas de Matemáticas con *DERIVE***, Distribuidora A.G.L.I., S.L., c/ Islas Marshall, 1, 28035 Madrid, Spain, 1994, ISBN 84-368-0759-6, phone (91) 373 66 40, fax (91) 373 24 00. Elaboradas por los profesores del Departamento de Matemática Aplicada de la E. U. I. (U. P. M.). Spanish language. 418 pages and a 3.5" diskette.

- Guzmán, M. de and Rubio, B.: **Análisis Matemático - 3 - sucesiones y series de funciones, números complejos, *DERIVE*, aplicaciones**, Ediciones Pirámide, S.A., Madrid, Spain, ISBN 84-368-0759-6. Spanish language.

- Koepf, Wolfram; Ben-Israel, Adi; and Gilbert, Bob: **Mathematik mit *DERIVE***, Verlag Vieweg, Faulbrunnenstrasse 13, D-65183 Wiesbaden, Germany, 1993, ISBN 3-528-06549-4, phone (06 11) 1602-18, fax (06 11) 1602-29. Neu ist die Zuhilfenahme des symbolischen Mathematikprogramms ***DERIVE*** als technisches Hilfsmittel zur praxisnahen und dennoch strengen Wissensvermittlung. Alle dargestellten Konzepte werden durch praktische Ubungen mit ***DERIVE*** unterstutzt. Dabei werden sowohl die numerischen als auch die graphischen Fahigkeiten solcher Systeme zum besseren Verstandnis der betrachteten Konzepte herangezogen. 394 pages.

- Koepf, Wolfram: **Höhere Analysis mit *DERIVE***, Vieweg-Verlag, Braunschweig-Wiesbaden, Germany, 1994, ISBN 3-528-06549-X.

- Kutzler, Bernhard: **Mathematik unterrichten mit DERIVE, Ein Leitfaden für Lehrer**, Addison-Wesley, 1995, ISBN 3-89319-860-1. bk teachware,

Softwarepark, A-4232 Hagenberg, Austria, phone +43-7236-6065, fax +43-7236-3769. Dieses Buck bespricht die Rolle von DERIVE sowohl im traditionellen wie auch im zukünftigen Mathematikunterricht. Praktische Übungsbeispiele, Tips und Tricks zu dem Produkt, die Beschreibung eines vom Hersteller nicht dokumentierten Produktmerkmales und ein Blick hinter die Kulissen von Computeralgebra-Systemen machen es zu einem Standardwerk für Lehrer. German language. 190 pages.

- Llorens Fuster, José Luis: **Introducción al Uso de *DERIVE*, Aplicaciones al Algebra Lineal y al Cálculo Infinitesimal**, Servicio de Publicaciones, Universidad Politécnica de Valencia, Departamento de Matemática Aplicada, Escuela Universitaria de Ingeniería Técnica Agrícola, Spain, 1993, 93.638, ISBN 84-7721-199-X. This book consists of two parts: 1) an introduction to using ***DERIVE***, and 2) applications of ***DERIVE***. Spanish language. 270 pages.

- Llorens Fuster, José Luis: **Aplicaciones de *DERIVE*, Análisis Matemático - I(Cálculo)**, Servicio de Publicaciones, Universidad Politécnica de Valencia, Departamento de Matemática Aplicada, Escuela Universitaria de Ingeniería Técnica Agrícola, Spain, 1993, 93.735, ISBN 84-7721-237-6, phone 96-387 70 12, fax 96-387 79 12. This book contains over 700 calculus problems solved using ***DERIVE***. Spanish language. 398 pages.

- Llorens Fuster, José Luis: **Aplicaciones de *DERIVE*, Algebra Lineal (Fundamentos)**, Servicio de Publicaciones, Universidad Politécnica de Valencia, Departamento de Matemática Aplicada, Escuela Universitaria de Ingeniería Técnica Agrícola, Spain, 1994, 93.753, ISBN 84-7721-255-4, phone 96-387 70 12, fax 96-387 79 12. This book discusses linear algebra including vectors and matrices, linear mappings, linear equations, diagonalization of endomorphisms, and Euclidean and related geometries. Spanish language. 345 pages.

- Lokar, Matija: ***DERIVE*, program za simbolicno racunanje**, Drustvo matematikov, fizikov in astronomov Slovenije, Zavod Republike Slovenije za solstvo in sport, Ljubljana, Slovenia, 1991. This book provides an introduction to ***DERIVE***. Slovenian language. 56 pages.

- Manara, A. and Perotti, A.: **Algebra lineare e geometria con *DERIVE***, McGraw-Hill, 1992, ISBN 88-386-0676-5. Italian language. 128 pages.

- Marlewski, Adam: ***DERIVE* Pomocnik matematyczny, Version 2**, Biblioteka Uzytkownika Mikrokomputerow, Wydawnictwo NAKOM, ul. Klasztorna 19, 61-799 Poznan, Poland, 1991, ISBN 83-85060-49-9. This book provides a comprehensive introduction to ***DERIVE***. Polish language. 355 pages.

- Mauve, R. and Moos, J.P.: **Mathematik mit *DERIVE* - Arbeitsblätter zur experimentellen Mathematik**, Dümmler Verlag, ISBN 3-427-45881-4. German language.

- Meer, A. van der: **Handleiding *DERIVE***, University of Twente, Faculty of Applied Mathematics, Enschede, 1995. Order via: U-Take, P.O. Box 217, 7500 AE Enschede, The Netherlands (syllabus nr. 621). Workbook for use in first and second year university courses in calculus (one and several variables), linear algebra, and differential equations. A dedicated utility file available on request (email: A.W.J.vanderMeer@math.utwente.nl). Dutch language. 74 pages.

- Njord, Stefan and Persson, Bengt: **Räkna med *DERIVE***, Studentlitteratur, Lund, 1994, ISBN 91-44-47271-4. Boken vänder sig till gymnasiekurs A+B och kräver inga förkunskaper i programmet ***DERIVE***. Swedish language. 180 pages.

- Paulogorrán, C. and Pérez, C.: **Cálculo Matemático con *DERIVE* para PC**, RA-MA Editorial, Carretera de Canillas, 144, 28043, Madrid, Spain, 1994, ISBN 84-7897-119-X. Spanish language. 390 pages.

- Rahi, Erkki: ***DERIVE***, Kustannus Ky Teknikus, 1. painos, Kirjapaino Grafia Oy, Turku, Finland, 1990, ISBN 951-95722-7-9. This books provides an introduction to ***DERIVE***. Finnish language. 52 pages.

- Rubio, Rafael Miñano: **¿Cómo Lo Hago con *DERIVE*? - Guía para empezar a usar *DERIVE***, Departamento de Publicaciones de la Escuela Universitaria de Informática, Universidad Politécnica de Madrid, 28031 Madrid, Spain, 1993, ISBN 84-87238-20-3. Spanish language. 51 pages.

- Ruffo, Giuseppe and Trifone, Anna: **Pascal e *DERIVE* per la matematica**, Zanichelli, 1993, ISBN 88-081-4700-2. Italian language. 272 pages.

- Sanchez, M. Angeles Martinez: **Problems Y Prácticas de Algebra**, Departamento de Publicaciones de la Escuela Universitaria de Informática, Universidad Politécnica de Madrid, 28031 Madrid, Spain, 1993, ISBN 84-87238-23-8. Spanish language. 32 pages.

- Scheu, Gunter: **Arbeitsbuch Computer-Algebra mit *DERIVE***, Ferd. Dummlers Verlag, Kaiserstr 31-37 (Dummlerhaus), 5300 Bonn 1, Germany, 1992, ISBN 3-427-45721-4. This book uses ***DERIVE*** to solve numerous unique examples from number theory and other applications. German language. 154 pages.

- Sjostrand, David and Wasen, Rolf: **Experiment med matematik. *DERIVE* i undervisningen**, David Sjostrand, Mejerivagen 31, 439 36 Onsala, Sweden, 1992, phone 0300/624 54. This book consists of numerous mathematical examples and experiments based on ***DERIVE***. Swedish language. 94 pages.

- Wasen, Rolf and Sjostrand, David: **Svensk Handledning med Ovningsuppgifter och Introduktion till Algebraisk Modellering och Datoralgebra: *DERIVE***, GraftiTex-Data, Storgatan 11, 590 40 Kisa, Sweden, 1991, phone 0494-100 77. This book provides an introduction to ***DERIVE***. Swedish language. 33 pages.

Menu Commands

The following is a summary of the menu commands available in ***DERIVE***. The commands are listed hierarchically in the same order as they appear in the menu. Each command is preceded by the number of the section in the ***DERIVE*** User Manual that describes it, and it is followed by a brief description of the command.

Algebra Window Commands

3.1	**Author**	enter a new expression on the author line
3.5	**Build**	combine expressions using operators and functions
	Calculus	
7.2	**Differentiate**	differentiate an expression
7.4	**Integrate**	integrate an expression
7.1	**Limit**	find the limit of an expression
7.6	**Product**	find the closed form product of an expression
7.5	**Sum**	find the closed form sum of an expression
7.3	**Taylor**	find the Taylor polynomial approximation of expression
8.2	**Vector**	generate a vector of values
	Declare	
4.12	**Function**	declare a function and optionally assign it a definition
	Variable	
4.11	**Value**	declare a variable and assign it a value
4.10	**Integer**	declare an integer-valued variable
4.10	**Real**	declare a real-valued variable
4.10	**Complex**	declare a complex-valued variable
4.10	**Nonscalar**	declare a variable nonscalar
8.1	**Matrix**	enter a matrix
8.1	**vectoR**	enter a vector
4.4	**Expand**	expand an expression wrt some or all variables
4.6	**Factor**	factor an expression wrt some or all variables
2.5	**Help**	get on-line help with
2.4	**Editing**	line editing commands
6	**Functions**	the pre-defined functions and constants
2.2	**Algebra**	algebra window commands
5.1	**2D-plot**	2D-plot window commands
5.2	**3D-plot**	3D-plot window commands
9	**Utility**	the functions defined in the utility files
2.14	**State**	the current state of the system control settings
2.5	**Return**	return to ***DERIVE***
3.3	**Jump**	move the highlight to a user-selected expression
4.14	**soLve**	solve a relation or a linear system of equations

	Manage	
3.12	**Annotate**	annotate an expression
4.5	**Branch**	select the principal, real, or any branch for simplification
6.1	**Exponential**	collect or expand exponentials
6.2	**Logarithm**	collect or expand logarithms
4.3	**Ordering**	determine the ordering used for variables
3.7	**Renumber**	sequentially renumber expressions in algebra window
4.8	**Substitute**	substitute a value for variables or subexpressions
6.3	**Trigonometry**	collect or expand trig functions and select angle mode
	Options	
	Color	
2.12	**Menu**	set the menu, message, and status line colors
2.12	**Work**	set the foreground and background colors of work area
2.12	**Display**	select graphics or text display mode
2.7	**Execute**	execute a DOS command
4.1	**Input**	set multi-character name, case-sensitive input, and default arrow key modes
2.3	**Mute**	turn on or off audible error beeps
3.9	**Notation**	set the notation used to display numbers
3.13	**Output**	set the display format and times operator
3.8	**Precision**	set the precision used for arithmetic operations
3.11	**Radix**	set the input and output radix bases
5.2	**Plot**	switch to a 2D-plot or a 3D-plot window
2.6	**Quit**	quit ***DERIVE***; return to DOS
3.7	**Remove**	delete a block of one or more expressions
3.2	**Simplify**	simplify an expression
	Transfer	
	Load	
2.9	**Derive**	load and display expressions from a math file
2.14	**State**	load control settings from a initialization file
2.9	**daTa**	load numeric data arrays from a data file
2.9	**Utility**	load definitions from a math file without displaying them
	Save	
2.8	**Derive**	save expressions in a ***DERIVE*** compatible file
2.8	**Basic**	save expressions in a Basic compatible file
2.8	**C**	save expressions in a C language compatible file
2.8	**Fortran**	save expressions in a Fortran compatible file
2.8	**Pascal**	save expressions in a Pascal compatible file
2.8	**Options**	select expressions to save, annotations, & file linelength
2.14	**State**	save the current control settings in an initialization file
2.9	**Merge**	merge expressions from a ***DERIVE*** file

	Clear	
2.9	**All**	clear all expressions, variable assignments, and functions
2.9	**Expressions**	clear expressions in the active algebra window
2.9	**Functions**	clear user-defined functions
2.9	**Variables**	clear user-assigned variable values and domains
2.9	**Demo**	run a demonstration from a ***DERIVE*** demonstration file
	Print	
	Expressions	
2.10	**Printer**	print expressions in algebra window
2.10	**File**	send text image of expressions to an ASCII file
2.10	**Options**	select print range, annotation, and character set
	Screen	
2.11	**Printer**	print screen image (= **Shift-F10** or **Shift-F9**)
2.11	**File**	save screen image in TIF file (= **Ctrl-F10** or **Ctrl-F9**)
2.11	**Options**	select print region, positioning, and TIF file name
2.10	**Layout**	set page size and margins
2.10	**Options**	select printer type, font size, orientation, and paper size
3.7	**Unremove**	recover last removed expressions
3.6	**moVe**	move a block of one or more expressions
	Window	
2.13	**Close**	close a window
2.13	**Designate**	redesignate the active window
2.13	**Flip**	flip overlaid windows (= **F2** key)
2.13	**Goto**	go to a given window number
2.13	**Next**	go to the next window number (= **F1** key)
2.13	**Open**	open a new window
2.13	**Previous**	go to the previous window number (= **Shift-F1**)
	Split	
2.13	**Horizontal**	split the active window horizontally
2.13	**Vertical**	split the active window vertically
3.10	**approX**	approximate an expression

2D-plot Window Commands

5.1.4	**Algebra**	switch to an algebra window
5.1.3	**Center**	center plot window near the cross
	Delete	
5.1.4	**All**	delete all expressions on the plot list
5.1.4	**Butlast**	delete all but the last expression on the plot list
5.1.4	**First**	delete the first expression on the plot list
5.1.4	**Last**	delete the last expression on the plot list
2.5	**Help**	(same as the algebra window's **Help** command)
5.1.1	**Move**	move the cross to the specified coordinates
	Options	
5.1.5	**Accuracy**	set the plot accuracy
	Color	
5.1.2	**Auto**	control the automatic changing of the plot color
5.1.2	**Plot**	set the plot and axes colors
2.12	**Menu**	set the menu, message, and status line colors
2.12	**Work**	set the background color of the plot area
2.12	**Display**	select graphics or text display mode
2.7	**Execute**	execute a DOS command
2.3	**Mute**	turn on or off audible error beeps
3.9	**Notation**	set the notation used to display numbers
3.8	**Precision**	set the precision used for arithmetic operations
3.11	**Radix**	set the input and output radix bases
5.1.6	**State**	set coordinates, follow mode, trace mode, and point size
5.1.1	**Plot**	plot the expression highlighted in the algebra window
2.6	**Quit**	quit ***DERIVE***; return to DOS
5.1.3	**Range**	set the plot range
5.1.3	**Scale**	set the plot scale (field of view)
	Transfer	
	Load	
2.14	**State**	load control settings from a initialization file
	Save	
2.14	**State**	save the current control settings in an initialization file
	Print	
	Screen	
2.11	**Printer**	print screen image (= **Shift-F10** or **Shift-F9**)
2.11	**File**	save screen image in TIF file (= **Ctrl-F10** or **Ctrl-F9**)
2.11	**Options**	select print region, positioning, and TIF file name
2.10	**Layout**	set page size and margins
2.10	**Options**	select printer type, font size, orientation, and paper size
2.13	**Window**	(same as the algebra window's **Window** command)
5.1.2	**aXes**	set the aspect ratio, and axes titles and labels
5.1.3	**Zoom**	zoom (adjust) the plot scale in or out

3D-plot Window Commands

5.2.4	**Algebra**	switch to an algebra window
5.2.2	**Center**	set the coordinates of the box center
5.2.3	**Eye**	set the coordinates of the viewer's eye
5.2.3	**Focal**	set the coordinates of the focal point
5.2.1	**Grids**	set the number of grid panels
5.2.1	**Hide**	control the display of hidden lines
5.2.2	**Length**	set the lengths of the sides of the transparent box
	Options	
	Color	
5.2.1	**Plot**	set the top, bottom, and axes colors
2.12	**Menu**	set the menu, message, and status line colors
2.12	**Work**	set the background color of the plot area
2.12	**Display**	select graphics or text display mode
2.7	**Execute**	execute a DOS command
2.3	**Mute**	turn on or off audible error beeps
3.9	**Notation**	set the notation used to display numbers
3.8	**Precision**	set the precision used for arithmetic operations
3.11	**Radix**	set the input and output radix bases
5.2.1	**Plot**	plot the expression highlighted in the algebra window
2.6	**Quit**	quit ***DERIVE***; return to DOS
	Transfer	
	Acrospin	
5.2.4	**Run**	run **AcroSpin** with an existing ACD data file
5.2.4	**Save**	save 3D plot points in an ACD data file
	Load	
2.14	**State**	load control settings from a initialization file
	Save	
2.14	**State**	save the current control settings in an initialization file
	Print	
	Screen	
2.11	**Printer**	print screen image (= **Shift-F10** or **Shift-F9**)
2.11	**File**	save screen image in TIF file (= **Ctrl-F10** or **Ctrl-F9**)
2.11	**Options**	select print region, positioning, and TIF file name
2.10	**Layout**	set page size and margins
2.10	**Options**	select printer type, font size, orientation, and paper size
2.13	**Window**	(same as the algebra window's **Window** command)
5.2.1	**aXes**	turn on or off the display of the axes
5.2.2	**Zoom**	zoom (adjust) the lengths of box sides in or out

Line Editing Commands

The following is a summary of the line editing commands available in ***DERIVE*** (see Section 2.4). The special key commands are entered by pressing a single key on the keyboard. The control key commands are entered by pressing the command letter while holding down the **Ctrl** key.

Special Key

Backspace — delete character left of cursor
Del — delete character at the cursor
Enter — enter line of text
Ctrl-Enter — enter and simplify line of text
Esc — abort edit and return to menu
Ins — toggle text input mode between insert and overwrite modes
F1 — get help on line editing commands or math functions
F3 — copy highlighted expression onto author line
F4 — copy highlighted expression onto author line and in parentheses
F6 — toggle arrow key mode between line-edit and subexpression modes

Cursor Movement

(Arrow, **Home**, and **End** keys are *not* available in subexpression mode)

Ctrl-S or **Left arrow** (←) — move cursor left a character
Ctrl-D or **Right arrow** (→) — move cursor right a character
Ctrl-A or **Ctrl-Left arrow** (Ctrl←) — move cursor left a word
Ctrl-F or **Ctrl-Right arrow** (Ctrl→) — move cursor right a word
Ctrl-], **Ctrl-Q S** or **Home** — move cursor to left end of edit line
Ctrl-, **Ctrl-Q D** or **End** — move cursor to right end of edit line

Text Deletion

Ctrl-H or **Backspace** — delete character left of cursor
Ctrl-G or **Del** — delete character at cursor
Ctrl-T — delete word beginning at cursor
Ctrl-Y — delete all text on edit line
Ctrl-Q Y — delete text on edit line at and to the right of cursor
Ctrl-Q H — delete text on edit line to the left of cursor

Miscellaneous Commands

Ctrl-M or **Enter** — enter line of text
Ctrl-J or **Ctrl-Enter** — enter and simplify line of text
Ctrl-[or **Esc** — abort edit and return to menu
Ctrl-V or **Ins** — toggle insert/overwrite modes
Ctrl-U — insert previous line of text
Ctrl-P — insert control or special character

Greek Letters

The Greek letters in the extended ASCII character set can be entered by typing their equivalent English name. On PC compatible computers, they can also be entered using an **Alt** key command as follows:

Greek letter	English name	Alt key	Greek letter	English name	Alt key
α	alpha	**Alt-A**	μ	mu	**Alt-M**
ß	beta	**Alt-B**	π	pi	**Alt-P**
Γ	gamma	**Alt-G**	σ	sigma	**Alt-S**
δ	delta	**Alt-D**	τ	tau	**Alt-T**
ϵ	epsilon	**Alt-N**	Φ	phi	**Alt-F**
Θ	theta	**Alt-H**	Ω	omega	**Alt-O**

Special Math Symbols

The math symbols in the extended ASCII character set can be entered by typing their equivalent ***DERIVE*** name. On PC compatible computers, some of them can also be entered using an **Alt** key command as follows:

Math symbol	*DERIVE* name	Alt key	Meaning
e	#e	**Alt-E**	base of natural logs
i	#i	**Alt-I**	square root of −1
π	pi	**Alt-P**	area of a unit circle
∞	inf	**Alt-0**	plus infinity
°	deg		radians per degree
$\pm$	"+−"	**Alt** −	plus-or-minus operator
$\downarrow$	SUB	**Alt-V**	subscript operator

Math symbol	*DERIVE* name	Alt key	Meaning
√	SQRT	Alt-Q	square root operator
Γ	GAMMA	Alt-G	gamma function
Σ	SUM		summation operator
Π	PRODUCT		product operator

Highlighting Commands

The following is a summary of the commands for highlighting expressions and subexpressions in an algebra window (see Section 3.3).

Expression Highlighting

Up arrow (↑) or **Ctrl-E** — move up one expression
Down arrow (↓) or **Ctrl-X** — move down one expression
PgUp or **Ctrl-R** — move up one half screen
PgDn or **Ctrl-C** — move down one half screen
Ctrl-PgUp or **Ctrl-Home** — move up to first expression
Ctrl-PgDn or **Ctrl-End** — move down to last expression

Sub-expression Highlighting

Right arrow (→) — move right an operand
Left arrow (←) — move left an operand
Home — move to leftmost operand
End — move to rightmost operand
Up arrow (↑) or **Ctrl-E** — move up one level
Down arrow (↓) or **Ctrl-X** — move down one level

Horizontal Scrolling

Ctrl-Right arrow (Ctrl→) or **Ctrl-F** — scroll expression left one half screen
Ctrl-Left arrow (Ctrl←) or **Ctrl-A** — scroll expression right one half screen

Function Key Commands

The following is a summary of the function key commands in ***DERIVE***.

Window Switching (see Section 2.13)

F1 — switch to the next window
Shift-F1 — switch to the previous window
F2 — flip to the next overlaid window
Shift-F2 — flip to the previous overlaid window

Line Editing (see Section 3.3)

F1 — get help on line editing commands or math functions while line editing
F3 — copy highlighted expression onto author line
F4 — copy highlighted expression onto author line and in parentheses
F6 — toggle arrow key mode between line-edit and subexpression modes

Display Mode Switching (see Section 2.12)

F5 — switch to the previous display mode (e.g. text or graphics)

Plot Window Commands (see Chapter 5)

F3 — toggle trace mode on and off (2D-plot window only)
F7 — magnifies image in the vertical or *y* direction
Shift-F7 — magnifies image in the horizontal or *x* direction
F8 — shrinks image in the vertical or *y* direction
Shift-F8 — shrinks image in the horizontal or *x* direction
F9 — magnifies image in both directions
F10 — shrinks image in both directions

Screen Image Printing and Saving (see Section 2.11)

Shift-F9 — sends graphics image of current algebra or plot window to printer
Shift-F10 — sends graphics image of entire ***DERIVE*** screen to printer
Ctrl-F9 — sends graphics image of current algebra or plot window to a TIF file
Ctrl-F10 — sends graphics image of entire ***DERIVE*** screen to a TIF file

Index

Abandon expressions (Y/N)? 25
Aborting commands 20, 77
Aborting computations 20
Aborting print-outs 32
ABS 52, 71, 175, 177, 208
Abscissa 242
Absolute value 52, 175, 177, 208
Acknowledgments iii
ACOS 172
ACOSH 174
ACOT 171
ACOTH 174
AcroSpin ii, 160, 161, 162
ACSC 172
ACSCH 174
Active selection field 20
Active window 38
Addition 44
Additional Resources 331
ADJOIN_ELEMENT 210, 233
ADJOINT 235
Adjugate 235
AI_SERIES 280
Airy functions 279, 280
Algebra 71
Algebra command 137
Algebra window 18, 38
Algebra Window Commands 341
ALGEBRA.DMO 3, 71
Algebraic expressions 75
Almost-linear equation 257
ALMOST_LIN 257
ALMOST_LIN_GEN 257
Alpha (α) 71
Alt key 15, 71
Alt-A (α) 71
Alt-B (ß) 71
Alt-D (δ) 71
Alt-E (ê) 164
Alt-F (Φ) 71
Alt-G (Γ) 71, 178
Alt-H (Θ) 71
Alt-I (î) 84, 177
Alt-M (μ) 71
Alt-N (ε) 71
Alt-O (Ω) 71
Alt-P (π) 65, 71, 167
Alt-Q (√) 52, 164
Alt-S (σ) 71
Alt-T (τ) 71
Alt-V (↓) 209
Alt-0 (∞) 98
Alt - (±) 98
Alternating series test 297
AND Boolean operator 119, 313
Angle mode 142, 143, 145
Angle-sum transformations 167
Angular measurement 169
Annotating Expressions 69
Annotation 43, 47, 50, 54
Annuity 184
Antiderivative 196
Antidifference 201
Antiquotient 203
APPEND 210
APPEND_COLUMNS 233
ApproX command 66, 163
APPROX.MTH 5, 272
APPROX_EIGENVECTOR 236
Approximate mode 60, 66
Approximate rational arithmetic 63
Approximating Irrational Numbers 66
Arbitrary constant 198
Arbitrary function 106
Arbitrary value (@n) 114, 115
ARC_LENGTH 248
Arccosecant 172
Arccosine 172
Arccotangent 171
ARCS 294
Arcsecant 172
Arcsine 171

Arctangent 171
AREA 248
AREA_CENTROID 249
AREA_INERTIA 249
AREA_OF_REVOLUTION 251
AREAY_OF_REVOLUTION 251
ARITH.DMO 3, 43
Arithmetic 43
Arithmetic mean 52, 180
ASCII character set 30, 34
ASEC 172
ASECH 174
ASIN 171
ASINH 174
Aspect ratio 127
Assigned variable 103
Assigning Values to Variables 103
Assignment operator (:=) 103, 106
ASSOCIATED_LEGENDRE_P 284
Asymptotic series 273
AT compatible computer 6
AT&T 6300 compatible adapter 35
ATAN 171
ATANH 174
Author command 22, 43, 137
Auto-scale mode 134, 135
AUTONOMOUS 261
Autonomous equations 260, 261
AUTONOMOUS_CONSERVATIVE 260
AVERAGE 52, 180
Axes 127, 149, 155, 156, 161, 287
AXes command 127, 155
Axes labels 127
Axes rows and columns 127
Axes titles 127

Back-accent operator (`) 215
Background color 31, 37
Backspace key 15, 20, 22
BAS (Basic) files 27
Basic programs 27
BAT (batch) files 41
Batch mode 29
BERNOULLI 256, 295
BERNOULLI_POLY 295
Bernoulli equation 256
Bernoulli numbers 295
Bernoulli polynomials 295
BERNOULLI_GEN 256
Bessel functions 279
BESSEL.MTH 5, 279
BESSEL_I 279
BESSEL_I_ASYMP 279
BESSEL_I_SERIES 279
BESSEL_J 279
BESSEL_J_ASYMP 279
BESSEL_J_SERIES 279
BESSEL_K 280
BESSEL_K_ASYMP 280
BESSEL_Y 279
BESSEL_Y_ASYMP 279
BESSEL_Y_SERIES 279
BETA 275
Beta (ß) 71
Beta function 275
BI_SERIES 280
Bibliography, annotated 333
Binary (base two) 67
Binomial coefficient 178
Binomial distribution function 276
Binomial probability density 276
BINOMIAL_DENSITY 276
BINOMIAL_DISTRIBUTION 276
Bit-wise logical operations 120
Bivariate expression 121, 153, 157, 160
Black and white images 32
Books, ***DERIVE*** 333
Boolean expressions 315
Boolean Operators 119
Bottom margin 31
Bounds, variable 100
Braces 44, 52, 205
Brackets 44, 52, 205, 325
Branches 84
Build command 54
Building Expressions 54
Built-up fractions 46
Bulletin board, electronic 332

C files 27
C programs 27
Calculus 189
Calculus Differentiate command 192, 196
Calculus Integrate command 196
Calculus Limit command 189
Calculus Product command 203
Calculus Sum command 201
Calculus Taylor command 194
Calculus Vector command 152, 207
CALCULUS.DMO 3, 189
Cannot find COMMAND.COM 26
Cartesian coordinates 222, 232, 237, 242, 247, 287
Case-insensitive mode 73
Case-sensitive mode 73
CATALAN 295
Catalan numbers 295
Catastrophic cancellation 63, 140, 184, 198
Cauchy principal value 199
CEILING 295
Center command 129, 130, 158
CENTER_OF_CURVATURE 242
CGA (Color Graphics Adapter) 35
Chain rule 193
Change of variable 197, 299
Character input mode 72, 227
Character set 30
Characteristic polynomial 220, 237
Characteristic vector 236
CHARPOLY 220
Chebychev polynomials 283, 308, 319
CHEBYCHEV_T 283
CHEBYCHEV_U 283
CHI 176
Chi-square distribution function 277
CHI_SQUARE 277
Choosing Complex Branches 84
CI 274
Circular functions 167
CLAIRAUT 257
Clairaut equation 257
CLAIRAUT_DIF 270
Clear variable value 103
Closed-form expression 196
Coefficient 78
COFACTOR 235
Color Graphics Adapter (CGA) 35
Color images 32
Color monitor 35
Color number 37, 127, 157
COMB 178
Combinations 178
Command line, DOS 29, 41
Command menu 19, 341
COMMAND.COM 26
Comments 73
Common denominator 88
Complementary error function 183
Complex conjugate 177, 228
Complex expression 84
Complex factoring 89
Complex infinity 98
Complex plane 84, 177
Complex Variable Functions 177
Compressed display format 70
Computer hardware 6
Conditional expressions 309
Conditions 309
CONE 291
Confluent hypergeometric function 281
CONJ 177
Conjunction 119, 313
Conservative equations 260
Constant coefficients 256, 270
Constants 71, 163
Continuation character 27
Continued fraction 306
Continuous plot line 142, 146
Contour plots 151
Control keys 22
Controlling Expression Format 70
Convergence tests 297
Conversion factors 4
Coordinate geometry matrix 222
Coordinate system 222

Coordinate vector 287, 288
Coprojection 289
Copy key 33
Copyright notice ii, 8, 18
COS 167
Cosecant 167
COSH 173
Cosine 167
Cosine integral 274
COT 167
Cotangent 167
COTH 173
Counting elements 310
Covariant metric tensor 238
COVARIANT_METRIC_TENSOR 238
CROSS 214
Cross mark 123, 124, 127, 130, 143
Cross product 214
CSC 167
CSCH 173
Ctrl key 15
Ctrl-A 22, 49, 124, 133, 144
Ctrl-C 48, 124, 133
Ctrl-D 22, 48, 124, 133, 144
Ctrl-E 48, 49, 123, 133, 138
Ctrl-End 48
Ctrl-Enter 47, 54, 146
Ctrl-F 22, 49, 60, 124, 125, 133, 144
Ctrl-F9 33
Ctrl-F10 33
Ctrl-G 22
Ctrl-H 22
Ctrl-Home 48
Ctrl-J 47, 54, 146
Ctrl-Left arrow 22, 49, 124, 133, 144
Ctrl-PgDn 48
Ctrl-PgUp 48
Ctrl-Q Backspace 22
Ctrl-Q D 22
Ctrl-Q H 22
Ctrl-Q S 22
Ctrl-Q Y 22
Ctrl-R 48, 124, 133
Ctrl-Right arrow 22, 49, 60, 124, 125, 133, 144
Ctrl-S 22, 48, 124, 133, 144
Ctrl-T 22
Ctrl-U 22
Ctrl-V 22
Ctrl-X 48, 123, 133, 138
Ctrl-Y 22
Ctrl-\ 22
Ctrl-] 22
Cumulative distribution function 183
CURL 224
Curl of a vector 224
Current state 40
Current window 18
Cursor 22, 347
CURVATURE 242
CYLINDER 291
Cylindrical coordinates 222, 237, 247
CYLINDRICAL_VOLUME 251

DAT (data) file 28
Data entry fields 21
Data matrix 147, 181
Data Matrix Plots 147
Data point 147
De Morgan's laws 316
Decimal (base ten) 67
Decimal approximation 65
Decimal point notation 64
Declare Function command 106
Declare Matrix command 205
Declare Variable command 99, 103
Declare vectoR command 137, 205
Declared variables 72, 103
Declaring Domains of Variables 99
DEF_INT_SUBST 300
Default domain 99
Defining Functions 106
Definite integral 196
Definite product 203
Definite sum 201
Deflated system 229
Deg (°) 167

Degree 169
Del key 15, 22
Delete All command 138
Delete Butlast command 138
Delete First command 138
Delete Last command 138
DELETE_ELEMENT 210
Delta (δ) 71
Demonstration files 3, 28, 69
DENOMINATOR 187
DERIVE 1
  books 333
  bulletin board 332
  copyright notice ii
  current state 40
  diskette 3, 35
  International Journal 331
  Internet mailing list 331
  License Agreement 3, 369
  Newsletter 331
  Problem Report Form 8, 369
  Registration Card 1
  screen 18
  starting 17
  subdirectory 9, 17
  system contents 3
  system requirements 6
  User Group 331
  User Manual 3
  Warranty, Limited 90 Day 3, 373
DERIVE XM 1, 3, 7, 216
DERIVE.EXE 3, 17
DERIVE.HLP 3, 25
DERIVE.INI 41
DERIVE.PIF 3
DERIVE.SYS 3
DeskjetC printer 30, 32
DET 215
Determinant 215, 220
DIF 192, 196
DIF_APPS.MTH 4, 242
DIF_DATA 240
DIF_NUM 240
DIF2_DATA 241
DIF2_NUM 240
Difference equation 269
Differential Vector Calculus 222
Differentiation 192, 242
Differentiation, numerical 240
Digamma function 275
Digits of precision 59
DILOG 286
Dilogarithm function 286
DIMENSION 208
Dimension of a vector 208
Direction keys 48, 209
DIRECTION_FIELD 267
Discontinuity 189, 198
Discrete points 142, 146, 147
Disjunction 119, 313
Disk operating system 6
Diskette Backup Procedure 9
Diskette drive 9
Display adapter 34, 36
Display modes 34
DIV 223
Divergence 223
Diverges 230, 304
Divide operator (/) 44
Division 44
DMO (demonstration) file 28, 69
Domain of variables 99
DOS Commands 26
DOS prompt 17, 25, 26
DOS system diskette 9, 17, 26
Dot, multiplication 46
Dot product 212, 325
Dotmatrix printer 30, 32
Double integrals 197
Double quote marks 73
Down arrow key 15, 48, 123, 133, 144
DSOLVE1 253
DSOLVE1_GEN 252
DSOLVE2 259
DSOLVE2_BV 260
DSOLVE2_IV 260
Dubious accuracy 198

e (base of natural logarithms) 164
E1 274
Editing expressions 50
EDPMI.OVL 3
EGA (Enhanced Graphics Adapter) 35
EI 273
EIGENVALUES 220, 236
Eigenvector 236
Electronic Bulletin Board 332
ELEMENT 209
Elementary row operations 218
Elementary transcendental exprs 75
Elliptic integrals 282
ELLIPTIC.MTH 5, 282
ELLIPTIC_E 282
ELLIPTIC_F 282
ELLIPTIC_PI 282
Email address ii, 369
EN 274
EN_ASYMP 274
End key 15, 22, 48
English unit system 4
ENGLISH.MTH 4, 105
Enhanced color monitor 35
Enhanced Graphics Adapter (EGA) 35
Enter a command 15
Enter key 15, 20, 43, 47
Entering equations 108
Entering Expressions 43
Entering Functions 52
Entering Infinite Values 98
Entering Relations 108
Entering Variables 71
Entering Vectors 205
Entry mode 22
Epsilon (ϵ) 71
Equations 108
ERF 183
ERFC 183
Error beep 21, 44, 77, 125, 126, 158
Error functions 183
Esc key 15, 20, 23, 77, 129, 133, 138, 154, 156, 161
EULER 266, 296
EULER_POLY 296
Euler numbers 296
Euler polynomials 296
Euler's constant 178
Euler's method 266
euler_gamma constant 178, 274
EXACT 255
Exact differential equation 255, 261
Exact mode 59
EXACT_EIGENVECTOR 236
EXACT_GEN 255
EXACT2 261
Exclusive or 119
EXP 164
EXP_INT.MTH 5, 273
Expand command 81, 92
EXPAND function 81, 92
Expanding Polynomials 81
Expansion point 194
Expansion variables 81
Exponential function 164
Exponential integral 273, 274
Exponential transformations 164
Exponentiation operator (^) 45
Expression Decomposition 187
Expression Highlighting Commands 349
Extended ASCII chars 30, 34, 71, 348
Extended memory 1, 7, 216
Extended rational expressions 75
EXTRACT_2_COLUMNS 267
Eye command 159

F-distribution function 277
F_DISTRIBUTION 277
F1 function key 24, 28, 39, 44
F2 function key 39
F3 function key 50, 76, 124, 125, 209
F4 function key 50, 209
F5 function key 36, 123
F6 function key 49, 73, 133
F7 function key 130, 158
F8 function key 130, 158
F9 function key 130, 158
F10 function key 130, 158

Factor command 87
FACTOR function 87
Factorial operator (!) 178
Factoring Expressions 87
Factorization variables 90
FACTORS 188
False 120
Fax number ii, 369
FIBONACCI 296
Fibonacci numbers 296, 307
Financial Functions 184
FIT 181, 302
Fixed-point form 230
Fixed-point iteration 230, 303, 304
FIXED_POINT 230
Floating point notation 28
FLOOR 176, 295
Floppy diskette drive 8
Focal command 160
Focal point 160
Follow cross mode 131
Font print size 30
FOR (Fortran) files 27
FORCE0 234
Fortran programs 27
Forward first difference 270
FOURIER 247
Fourier series 247
Fractional-part 176
Fractions 59
Framing 2D Plots 129
Framing 3D Plots 157
Freeware warning 18
Fresnel integrals 278
FRESNEL.MTH 5, 278
FRESNEL_COS 278
FRESNEL_COS_ASYMP 278
FRESNEL_COS_SERIES 278
FRESNEL_SIN 278
FRESNEL_SIN_ASYMP 278
FRESNEL_SIN_SERIES 278
FUN_LIN_CCF 256
FUN_LIN_CCF_GEN 257
Function Key Commands 350
FUNCTION.DMO 3, 163
Functions 52, 106, 163
Fundamental physical constants 4
Future value 184
FVAL 184

GAMMA (Γ) 71, 178
Gamma function 178, 275
Gauss hypergeometric function 281
Gauss-Jordan elimination 233
Gaussian distribution 183
Gaussian elimination 116, 233
GCD 52, 186
GEGENBAUER_C 285
GEN_HOM 256
GEN_HOM_GEN 256
Generalized error function 183
Generalized homogenous 256
GENERALIZED_LAGUERRE 284
Generating Matrices 205, 207
Generating Tables 207
Generating Vectors 207
GEOMETRIC1 270
GEOMETRY_MATRIX 239
GFTABLE.COM 326
Goodness of fit 302
GOODNESS_OF_FIT 302
GRAD 222
Gradient 222
GRAFTABL.COM 326
GRAPHICS.COM 26, 329
GRAPHICS.MTH 5, 121, 147, 287
Greatest common divisor (gcd) 52, 186
Greek letters 71, 348
Grid lines 156, 159, 161
Grid panels 156
Grid topology 293
Grids command 156

Hard disk drive 8
Hard Disk Procedure 8
Heaviside step function 175
Helix space curve 288
Help command 24

Help system 44
Hercules compatible graphics adapter 35
HERCULES.COM 3, 26, 35
Hermite normal form 218
HERMITE_H 284
HERMITE_HE 284
Hermitian matrices 220
Hexadecimal (base sixteen) 67
Hexagamma function 275
Hidden lines 156, 289
Hide command 156
High resolution 34
Highlight 20, 43
Highlighting Expressions 48
Home key 15, 22, 48
HOMOGENEOUS 254
Homogeneous expression 254
HOMOGENEOUS_GEN 255
Horizontal scrolling 49
HORIZONTALS 293
Horn of Gabriel 251
Hornerized form 141
HP 95LX Palmtop computer 6
HP 100LX Palmtop computer 6
HP 200LX Palmtop computer 6
Hurwitz generalized zeta function 286
HURWITZ_ZETA 286
Hyperbolic cosecant 173
Hyperbolic cosine 173
Hyperbolic cotangent 173
Hyperbolic functions 173
Hyperbolic secant 173
Hyperbolic sine 173
Hyperbolic tangent 173
HYPERGEO.MTH 5, 281
HYPERGEOMETRIC 281
Hypergeometric distributions 276
Hypergeometric functions 281
Hypergeometric probability density 276
HYPERGEOMETRIC_DENSITY 276
HYPERGEOMETRIC_DISTRIBUTION 276
HYPERGEOMETRIC_SERIES 281

i (imaginary unit) 84, 177
I_ 232
Identity matrix 208
IDENTITY_MATRIX 208
IF 309
IF expressions 309
IM 177
Imaginary infinity 98
Imaginary part 177
Imaginary unit, $\sqrt{(-1)}$ 84, 177
IMP Boolean operator 119
IMP_CENTER_OF_CURVATURE 245
IMP_CURVATURE 245
IMP_DIF 245, 253
IMP_OSCULATING_CIRCLE 246
IMP_PERPENDICULAR 245
IMP_TANGENT 245
Implication 119
Implicit derivative 245
Implicit multiplication 45
Implicit plots 125, 149, 150
Improper integrals 198
Inapplicable form 253-257, 259, 261
Incomplete beta function 276
Incomplete gamma function 275
INCOMPLETE_BETA 276
INCOMPLETE_GAMMA 275
INCOMPLETE_GAMMA_SERIES 275
Indicator function 176
Inequalities 108
Inf (∞) 98
Infinite series 297
Infinity 190
INI (initialization) file 40
Inner product 212
Input base 67
Input mode 72
Ins key 15, 22
Insert mode 18, 22
INSERT_ELEMENT 211
INT 196
INT_APPS.MTH 4, 247
INT_DATA 241
INT_PARTS 299
INT_SUBST 299

Integer-part 176
Integral Vector Calculus 225
INTEGRATING_FACTOR 255
INTEGRATING_FACTOR_GEN 255
Integration 196, 247
Integration by parts 197, 299
Integration by substitution 299
Integration, numerical 198, 241
Intel 386 microprocessor 7
Intel 486 microprocessor 7
Intel 8086 microprocessor 6
Interest rate 184
International ***DERIVE*** Journal 331
Internet Mailing List 331
Internet web page ii, 331, 369
Interval, restricted 99
INVERSE 300
Inverse hyperbolic cosecant 174
Inverse hyperbolic cosine 174
Inverse hyperbolic cotangent 174
Inverse hyperbolic functions 174
Inverse hyperbolic secant 174
Inverse hyperbolic sine 174
Inverse hyperbolic tangent 174
Inverse iteration method 236
Inverse, matrix 216
Inverse of expression 231, 304
Inverse Trigonometric Functions 171
Irrational numbers 59
ISOMETRIC 288
Isometric projection 287
ISOMETRICS 289
Iterated definite integrals 197
Iteration functions 303

J_ 232
JACOBI_P 284
JACOBIAN 238
Jacobian matrix 237
Jump command 48

K_ 232
Key Notation and Entry 15
KRONECKER 232
Kronecker delta function 232
KUMMER 281
KUMMER_SERIES 281

Label number 43, 48, 51
LAN (Local Area Network) 11
LAN license 11
Landscape orientiation 31
LAPLACE 247
Laplace transform 247
LAPLACIAN 223
Laserjet printer 30, 32
LCM 52, 186
Least common multiple (lcm) 52, 186
Least squares fit 181
Left arrow key 15, 22, 48, 124, 133, 138, 144
Left margin 31
Left Riemann sum 299
Left-hand limit 190
LEFT_RIEMANN 298
Legal size paper 31
LEGENDRE_P 283
Length command 158
Length, vector 208
Lerch transcendent function 286
LERCH_PHI 286
LHS 188
LI 274
License Agreement, ***DERIVE*** 3
LIM 189
LIM2 298
Limit point 189
Limits 189
LIN_FRAC 257
LIN_FRAC_GEN 257
LIN1_DIFFERENCE 269
LIN2_CCF 270
LIN2_CCF_BV 271
Line Editing 22, 347
Line Editing Commands 347
Line length 27
Line-edit mode 73
Linear equation 108, 110, 115, 218, 228

Linear factors 89
Linear fractional equation 257
Linear-geometric recurrences 270
LINEAR1 254
LINEAR1_GEN 254
LIOUVILLE 260
Liouville differential equation 260
LN 166
Loading Expressions 28
Local Area Network (LAN) 11
LOG 166
Log-log plots 136
Logarithmic Functions 166
Logarithmic integral 274
Logarithmic transformations 166
Logical Operators 313
LPT printer ports 13, 31, 41

Magnitude 84, 177, 208
Mailing list, Internet 331
Manage Annotate command 69
Manage Branch command 85
Manage Exponential command 164
Manage Logarithm command 166
Manage Ordering command 79
Manage Renumber command 58
Manage Substitute command 94, 191
Manage Trigonometry command 112, 142, 167
Margins 31
MathPORT utility program 330
MATPROD 234
Matrix 205
Matrix Algebra 221
Matrix equation 217, 218
Matrix inverse 216
Matrix Operations 215
Matrix rank 218, 235
Matrix transpose 215
MATRIX.DMO 4, 205
MAX 175
MCGA (Multi-color Graphics Array) 35
MDA (Monochrome Display Adapter) 35
Medium resolution 34
Memory, computer 1, 6
Memory Full 77, 156
Memory space 18
Memory Usage, ***DERIVE XM*** 14
Menu Commands 341
Menu options 19
Menu System 20
Menu title 19
MERSENNE 296
Message line 19
Metric unit system 4
METRIC.MTH 4, 105
Microsoft Windows 3, 6, 7
MIN 175
MINOR 233
Minus infinity ($-\infty$) 98, 190
Minus operator (–) 44
MISC.MTH 5, 297
Mixed mode 61
Mixed notation 64
Mixed partial derivatives 192
MOD 176
MODS 176
Modulo 176
Modulus 177
Monic differential equation 254
Monochrome Display Adapter (MDA) 35
Monochrome monitor 35
MONOMIAL_TEST 256
Monotonic sequence 314
Monte Carlo primality test 186
MoVe command 56, 123, 125
Moving Expressions 56
MS-DOS 6, 7
MTH (math) files 27
mu (μ) 71
muLISP ii, 321, 331
Multi-color Graphics Array (MCGA) 35
Multiple plots 137, 144, 146
Multiple-angle transformations 167
Multiplication 44, 70
Multivariate limit 298
Multivariate polynomials 75

Natural logarithm 166
NEC PC-9801 Personal Computer 6
Negation 44, 119
Netware 11
Newsletter, ***DERIVE*** 331
Newton's method 228, 305
NEWTONS 228
NEXT_PRIME 186
No elementary integral 197
No solutions found 114, 115
Nonstrict bound 101
NORMAL 183
Normal display format 70
Normal distribution function 183
Normal form 75, 78
NORMAL_LINE 246
NOT Boolean operator 119, 315
Notation style 27, 64
Novell Netware LAN 11
NPER 184
Nth order antiderivative 196
Nth order partial derivative 192
NTH_PRIME 295, 307
Number Theory Functions 186
NUMBER 186
NUMBER.MTH 5, 295
NUMERATOR 187
Numeric coefficient 78
NUMERIC.MTH 4, 240
Numerical differentiation 240
Numerical integration 198, 241, 298

Octal (base eight) 67
ODE_APPR.MTH 4, 263
ODE1.MTH 4, 252
ODE2.MTH 4, 259
Omega (Ω) 71
Operator precedence 44
Operators 44
Options Accuracy command 139, 149
Options Color Auto command 127, 139, 148
Options Color Menu command 37
Options Color Plot command 127, 139, 156
Options Color Work command 37
Options Display command 34, 123
Options Execute command 26
Options Input command 72, 73, 227
Options Mute command 21
Options Notation command 64, 124
Options Output command 70
Options Precision command 59, 140
Options Radix command 67
Options State command 124, 125, 131, 142, 148
OR Boolean operator 119, 314
Order list 78
Ordering Variables 78
Ordinate 242
ORTH_POL.MTH 5, 283
Orthogonal polynomials 283
OS/2, IBM operating system 6, 7
OSCULATING_CIRCLE 243
OUTER 232
Outer product 232
Output base 67
Overlay files 11
Overwrite mode 18, 22

PADE 272
Pade rational approximation 272
Page length 31
Page width 31
Paper size 31
PARA_ARC_LENGTH 248
PARA_CENTER_OF_CURVATURE 243
PARA_CURVATURE 243
PARA_DIF 243
PARA_OSCULATING_CIRCLE 244
PARA_PERPENDICULAR 244
PARA_TANGENT 244
Parametric plot 145, 146, 147
Parametric surfaces 287
Parentheses 44, 52, 205, 325
Partial derivative 192
Partial Fraction Expansion 92

PAS (Pascal) files 27
Pascal programs 27
Pascal's triangle 178
Payment 184
PC compatible computer 6
PC-9801 compatible computer 6
PC-DOS 6
PC-Junior graphics adapter 35
Pentagamma function 275
Pentium microprocessor 7
Percent operator (%) 46
Period infix operator (.) 212
PERM 178
Permutations 178
PERPENDICULAR 242
Personal computer 6
Personal System/2 Computer 6
PgDn key 15, 48, 124, 133
PgUp key 15, 48, 124, 133
PHASE 177
Phase angle 84, 177
Phase-plane plot 267
Phi (Φ) 71
Physical constants 105
PHYSICAL.MTH 4, 105
Pi (Π) 204
Pi (π) 65, 71, 167
PICARD 263, 264
Piecewise Continuous Functions 175
Piecewise definitions 309
PIVOT 234
Pivot row 234
Pixel 34
Plain printer 30
Plot accuracy 139, 149
Plot color 127, 139, 156
Plot command 122, 127, 137, 138, 142, 145, 147, 154
Plot example files 4
Plot range 129, 132, 133, 137
Plot scale 127, 129, 130, 133, 137
PLOT2D.MTH 4, 127
PLOT3D.MTH 4, 157
PLOTPARA.MTH 4, 147
Plotting 121
Plotting algorithm 139, 150
Plotting complex expressions 291
Plotting parametric surfaces 289
Plotting space curves 288
Plotting speed 139, 150
Plotting vectors 137
Plus infinity (∞) 98, 190
Plus operator (+) 44
Plus-or-minus infinity (±∞) 98, 191
Plus-or-minus one (±1) 191
PMT 184
POCHHAMMER 275
Pochhammer's symbol 275
Poisson distribution function 276
Poisson probability density 276
POISSON_DENSITY 276
POISSON_DISTRIBUTION 276
Polar coordinates 142, 143, 151
Polar moment 249
Polar plot mode 143, 145, 149, 151
Polar-cylindrical coordinates 222, 237
POLAR_ARC_LENGTH 248
POLAR_CENTER_OF_CURVATURE 244
POLAR_CURVATURE 244
POLAR_DIF 244
POLAR_OSCULATING_CIRCLE 244
POLAR_PERPENDICULAR 244
POLAR_TANGENT 244
POLY_COEFF 301
POLY_DEGREE 301
POLY_GCD 188
POLY_INTERPOLATE 302
POLY_INTERPOLATE_EXPRESSION 302
POLYGAMMA 275
Polygamma function 275
POLYLOG 286
Polylogarithm function 286
Polynomial coefficient 301
Polynomial degree 301
Polynomial expansion 81
Portrait orientation 31
POTENTIAL 225
Power mod algorithm 176, 295

POWER_MOD 295
Precedence 45
Precision 59
Present value 184
Primality test 186
Primary variable 78
PRIME 186
Prime factors 87
Prime numbers 87, 186, 295, 307
Principal branch 85
Print annotations 30
Print orientation 31
Print range 30
Printer ports 13, 31, 41
Printer not ready 30
Printer, system 30, 32
Printer type 30, 32, 41
Printing Expressions 30
Printing Questions 327
Printing Screen Images 32
PROBABIL.MTH 5, 275
Probability Functions 178, 275
Problem Report Form 8, 369
PRODUCT 203
Products 203
Program files 3
Programming 303
Proper rational expression 92
PROVE_PRODUCT 301
PROVE_SUM 300
PRT (print) files 30
PS/2 6
Pseudo-random number generator 179, 301
PSI 275
Psi function 275
Purified solution 229
PVAL 184

Quasi-linear equation 257
Question mark 98
Quit command 25
Quote mark 50, 76
QUOTIENT 187
Quotient polynomial 93

Radian 169
Radians per degree 167
Radical factoring 89
Radicals 59
Radius of gyration 249
Radix base 67
Raised exponents 46
RAM card, PCMCIA 6
RANDOM 179
Random number generator 179, 301
Random number seed 179
RANDOM_MATRIX 301
RANDOM_POLY 301
Random_sign 301
RANDOM_VECTOR 301
Range box 132, 133
Range command 129, 132
RANK 235
Rank, matrix 218
RATE 184
Ratio test 297
RATIO_TEST 297
Rational approximation 65
Rational arithmetic 59
Rational expressions 75
Rational factoring 88
Rational notation 64
Ratios 59
Ray plots 292
RAYS 293
RE 177
README file 3
Real expression 84
Real part 177
Rectangular coordinates 134, 142
Rectangular form 84, 163
Rectangular plot mode 145, 149
RECUREQN.MTH 5, 269
Recurrence equation 269
RECURRENCE1 269
Recursion 317
Recursive case 317

Recursive definitions 317
Recursive Functions 317
Registration Card 1
Relational operators 108, 309
Relations 108, 309
Remainder 176, 187
Remainder polynomial 93
Removable singularity 97
Remove command 57
Removing expressions 57
Renumbering expressions 58
REPLACE_ELEMENT 211
Resident program 26, 33, 35, 329
Reverse video 43
REVERSE_VECTOR 211
RHS 188
Riemann integral 199
Riemann sum 298
Riemann zeta function 5, 183, 286
Right arrow key 15, 22, 48, 124, 133, 138, 144
Right margin 31
Right Riemann sum 299
Right-hand limit 189
RK 266
RMS 180
Root mean square 180
ROTATE_X 288
ROTATE_Y 288
ROTATE_Z 289
Rotating coordinate vectors 288
ROUND 295
Roundoff errors 62
Row echelon form 218, 233
ROW_REDUCE 218
Runge-Kutta method 266

Sample standard deviation 181
Sample variance 180
Saving Expressions 27
Scalar potential 225
Scalar product 212
Scale command 129, 130, 134
SCALE_ELEMENT 233
Scientific notation 46, 64, 66
Screen capture programs 33, 35, 329
Scrolling, horizontal 49
SEC 167
Secant 167
SECH 173
Secondary variable 78
SELECT 211
Selection fields 20
Semi-log plots 136
SEPARABLE 254
SEPARABLE_GEN 254
SGN 311
Shareware warning 18
Shift key 15, 20
Shift-F1 39
Shift-F2 39
Shift-F7 130, 158
Shift-F8 130, 158
Shift-F9 32
Shift-F10 32
Shift-PrtSc 33, 329
Shift-Tab 20
SI 274
Sigma (σ) 71
Sigma (Σ) 201
SIGN 175, 177, 311
Significant digits 60
Simplification Questions 324
Simplify command 47, 74, 108
Simplifying Expressions 47, 74
Simpson's rule 198
SIN 167
Sine 167
Sine integral 274
Singular matrix 216
Singular solution 257
Singular system of equations 115, 218
Singularity 198
SINH 173
SMOOTH_COLUMN 240
SMOOTH_VECTOR 240
Soft Warehouse Europe GmbH ii, 369
Soft Warehouse, Inc. i, ii, 321, 369

Soft Warehouse web page ii, 331, 369
Solution variables 115
SoLve command 112, 115, 150
SOLVE.MTH 4, 228
Solving Equations and Relations 112
Solving Systems of Equations 115, 218
Space bar 15, 20
Space curves 287
Special Math Symbols 348
SPHERE 289
Spherical coordinates 222, 223, 237, 247, 289
SPHERICAL_BESSEL_J 280
SPHERICAL_BESSEL_Y 280
SPHERICAL_VOLUME 250
Spurious solutions 111
SQRT (√) 52, 71, 164
Square matrix 215
Square root 52, 164
Square-free factoring 88
SQUARE_WAVE 295
Standard deviation 181, 302
Starting ***DERIVE*** 17
Statistical Functions 180
Status line 18, 123, 156
STDEV 181
STEP 175
Step size 207
Strict bound 100
STUDENT 277
Student's t-distribution function 277
SUB 209
Sub-command menu 20
Subexpression mode 73
Subscript notation 209
Substituting Values 94
SUBTRACT_ELEMENTS 233
Subtraction 44
Sufficiently simple expression 75
SUM 201
Sums 201
Super VGA 35
Super-proper rational expression 92
SUPPORT.PAS 5, 27
Surds 59
SURFACE_AREA 249
SWAP_ELEMENTS 233
Symmetric matrix 220, 237, 249, 250
Syntax error 44
System Contents 3
System diskette 9
System of equations 115
System Requirements 6

Tab key 15, 20
Table, making 208
Table of Contents iv
Tagged Image Format (TIF) files 33
TAN 167
TANGENT 167, 242
TANGENT_PLANE 246
TANH 173
Tau (τ) 71
TAYLOR 194
Taylor Polynomial Approximations 194
Taylor series 272
TAYLOR_INVERSE 231
TAYLOR_ODE1 263
TAYLOR_ODE2 265
TAYLOR_ODES 265
TAYLOR_SOLVE 231
Technical support 1, 369
Telescoping products 204
Telescoping sums 202
Terminal case 317
TERMS 188
Tetragamma function 275
TEX files 330
Text mode 34
Theta (Θ) 71, 143
Three dimensional space 153
TIF (Tagged Image Format) files 33, 327, 330
Tilde (~) 27
Time in centiseconds 179
Time of payment 184
Times operator (*) 44, 70
Title page i

Token 22
Top margin 31
TORUS 290
Toshiba 3100 compatible graphics 35
TRACE 215
Trace mode 124, 138, 143
Trace of a square matrix 215
Tracing 124, 125, 126, 131, 143, 147
Trademark notice ii
Transcendental equation solve 303
Transfer Acrospin Run command 162
Transfer Acrospin Save command 160
Transfer Clear All command 29
Transfer Clear Expressions command 29, 57
Transfer Clear Functions command 29, 106
Transfer Clear Variables command 29, 99, 103
Transfer Demo command 28
Transfer Load daTa command 28
Transfer Load Derive command 28
Transfer Load State command 40
Transfer Load Utility command 28
Transfer Merge command 28
Transfer Print Expressions Options 30
Transfer Print Expressions File 30
Transfer Print Expressions Printer 30
Transfer Print Layout 30
Transfer Print Layout command 31
Transfer Print Options command 30, 32
Transfer Print Screen File command 33
Transfer Print Screen Options 33
Transfer Print Screen Printer 32
Transfer Save Basic command 27
Transfer Save C command 27
Transfer Save Derive command 27
Transfer Save Fortran command 27
Transfer Save Options command 27
Transfer Save Pascal command 27
Transfer Save State command 40
Transparent box 157, 159
Transpose, matrix 215
Transpose operator (`) 215
TRIG.DMO 3, 163
Trigonometric Functions 167
Trigonometric transformations 167
Trimetric projection 287
Triple integrals 197
Trivial factoring 88
True 120
Truth tables 120
Truth value 309
Truth value constants 120
TRUTH_TABLE 120
Two's-complement 120

Unassigned variable 103
Underscores 73
unit_circle constant 98, 177
Unit circle 167, 177
Unit conversion factors 4
Unit step function 175
Unit vectors 232
Units conversion 105
Univariate expression 121, 122, 134, 142, 145, 154
Univariate polynomials 75
Unremove command 57
Unremoving expressions 57
Up arrow key 15, 48, 49, 123, 133, 138, 144
Upgrade discounts 1
User contributed files 5
User group, ***DERIVE*** 331
User Interface Questions 323
User-defined function 106
Utility files 4, 28, 227

VAR 180
Variable names 71
Variable order list 81, 90
Variables 71, 188
Variance 180
Variance-ratio distribution function 277
VECTOR 137, 207
Vector Algebra 221
Vector length 208

Vector magnitude 208
Vector Manipulation Functions 209
Vector Operations 212
Vector potential 226
VECTOR.MTH 4, 223, 232
VECTOR_POTENTIAL 226
Vectors 205
Version number 8, 18, 369
Vertical bars 175
VGA (Video Graphics Array) 35
Video Graphics Array (VGA) 35
VOLUME 249
VOLUME_CENTROID 250
VOLUME_INERTIA 250
VOLUME_OF_REVOLUTION 251
VOLUMEY_OF_REVOLUTION 251

Warranty, ***DERIVE*** 3, 373
Web page, Soft Warehouse ii, 331, 369
WEBER_D 284
Window Close command 38
Window Commands 38
Window Designate command 38, 131, 154
Window Flip command 39
Window Goto command 39
Window Next command 39
Window number 38
Window Open command 38
Window Previous command 39
Window Split Horizontal command 38
Window Split Vertical command 38
Windows, Microsoft 6, 7
Windows NT, Microsoft 6, 7
Wire frame grid 153
Word input mode 72, 227
Working copy 8
Write-protect 8

XOR Boolean operator 119
x-*y* plane 171

Yen (¥) character 215

Zero polynomial 301
Zeros of an expression 112
ZETA 183
ZETA.MTH 5, 286
Zoom command 129, 130, 158

2D Plot Questions 325
2D-plot window 121, 122, 123, 127, 138, 145, 157, 159
2D-plot Window Commands 344

3D coordinate vector 287
3D Plot Questions 326
3D-plot window 153, 160
3D-plot Window Commands 345

ê (#e) base of natural logarithms 164
î (#i) imaginary unit 84, 177
∞ (inf) infinity 98
? unknown quantity 98
π (pi) area of unit circle 167
° (deg) radians per degree 167
Θ theta 71, 143, 149

@n arbitrary value 73, 114, 115
#n label number 51

+ plus operator 44
– minus operator 44
* times operator 44
/ divide operator 44
± plus-or-minus operator 98
^ exponentiation operator 45, 216
% percent operator 46
√ square root operator 52, 164
! factorial operator 178
↓ (sub) subscript operator 209
. dot product operator 212
` transpose operator 215
¥ transpose operator 215
:= assignment operator 103, 106

= equal operator 108
/= nonequal operator 108

< less than operator 108
> greater than operator 108
≤ (< =) less than or equal oper 108
≥ (> =) greater than or equal oper 108

Γ Gamma function 178
Σ Sigma summation notation 201
Π Pi product notation 203

| absolute value delimiter 52, 175
(left parenthesis 44
) right parenthesis 44
[left bracket 44, 52, 205
] right bracket 44, 52, 205
{ left brace 44, 52
} right brace 44, 52

DERIVE Problem Report Form

If you encounter a software bug, deficiency, or documentation error in the ***DERIVE*** or ***DERIVE XM*** program, you can submit a problem report directly to **Soft Warehouse** via our Internet web page at http://www.derive.com. We will try to resolve the problem and provide you with a correction.

Alternatively, fill out a copy of this form and mail it to **Soft Warehouse, Inc.**, 3660 Waialae Avenue, Suite 304, Honolulu, Hawaii, 96816-3236, U.S.A.; email it to swh@aloha.com; or fax it to (808) 735-1105. If you are in Europe, mail it to **Soft Warehouse Europe GmbH**, Softwarepark, A-4232 Hagenberg, AUSTRIA; email it to swhe@swp.co.at; or fax it to +43 7236 329780.

Name _____ Phone _____

Company/Institution _____ Fax _____

Division/Department _____ Email _____

Street _____

City _____ State ____ Zip _____

Country _____ Date _____

Computer processor/name _____ Memory size _____

Peripherals _____ Diskette size _____

Version number displayed when the following programs are loaded:

DERIVE __ . ___ ***DERIVE XM*** __ . ___

MS-DOS __ . ___ **Windows** __ . ___

Describe the problem and how to reproduce it (attach listing if applicable), and give your diagnosis if any. Use the reverse side of this form if necessary:

_____

_____

_____

_____

_____

_____

Problem description continued:

Soft Warehouse, Inc.

End-User License Agreement

READ THIS LICENSE AGREEMENT CAREFULLY BEFORE OPENING THIS PACKAGE. BY OPENING THIS PACKAGE, YOU INDICATE YOUR ACCEPTANCE OF THE FOLLOWING TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, PROMPTLY RETURN THIS UNOPENED PACKAGE TO THE PLACE OF PURCHASE FOR A REFUND.

License Grant: Soft Warehouse, Inc. ("SWH") grants you a nonexclusive license to use (1) the enclosed Software (the "Software") and (2) the contents of the Manual (the Software and the Manual are the "Product") on the following terms and conditions:

Ownership: SWH owns all title and rights, including copyrights and trade secret rights, in the Product and copies or modifications of the Product recorded on any media, although you own the media. This Agreement is *not* a sale of the Product or of any copy of the Product.

Copy Restrictions: The Product is protected by copyright law and by international treaty provisions. By law, you may either (a) make a single copy of the Software solely for backup or archival purposes or (b) transfer the Software to a single hard disk provided you keep the original solely for backup or archival purposes. Other copies are not authorized, except as follows: (a) as permitted by the terms, in writing, of an accompanying Site License Agreement or LAN License Agreement, if any, which terms are hereby incorporated by reference in this Agreement; or (b) you may store a copy on two computers (such as a desktop and a laptop or palmtop), *provided that you are the only user at any time*. You may not otherwise transmit or download the Software electronically between computers. Unauthorized copying or modification of the Product in any form is expressly forbidden. You can be held legally responsible for any copyright infringement that you cause or encourage. You must include the SWH copyright and trademark notices on all copies of the Software.

Permitted Use: This is a one-person license for your sole and exclusive individual use, subject to the above Copy Restrictions. You may not decompile, disassemble, reverse engineer, adapt, modify, translate, or create derivative works from the Product.

Transfer Restrictions: You may not sublicense, lend, lease, or rent the Product or any portion of the Product. You may permanently transfer your license provided that (1) you do not keep any copy of the Product, including revisions or updates, in any form, (2) the transferee accepts all the provisions of this Agreement, and (3) you promptly notify SWH in writing of the name and address of the transferee.

Term: This license will terminate automatically if you do not comply with any provision of this Agreement. Upon termination, you shall destroy all copies of the Product. All provisions of this Agreement relating to disclaimers of warranties, limitation of liability, remedies, or damages, and SWH's proprietary rights shall survive termination.

Costs of Litigation: If either party brings any action regarding this Agreement against the other, the prevailing party will be entitled to recover, in addition to any other relief granted, reasonable attorney's fees and expenses of litigation.

No Waiver: The failure of either party to enforce any rights granted under this Agreement or to take action in the event of any breach shall not be deemed a waiver as to subsequent enforcement of rights or subsequent actions in the event of future breaches.

Law: If any provision of this Agreement is invalid as a matter of law, every other provision of this Agreement shall remain in full force and effect. This Agreement is governed by the laws of the State of Hawaii, United States of America.

Entire Agreement: This Agreement is the entire agreement between you and SWH with respect to the Product and supersedes any prior proposal, representation, understanding, or agreement.

U.S. GOVERNMENT RESTRICTED RIGHTS: The Product is Restricted Computer Software, as defined in Federal Acquisition Regulations 48 CFR 52.227-19(a)(1992), and is Commercial Computer Software, as defined in DFARS 48 CFR 252.227-7013(a)(1)(1992). The Product is commercial computer software and related documentation developed at private expense and not in the public domain.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software Restricted Rights clause at 48 CFR 52.227-19, as applicable. The Contractor/manufacturer is Soft Warehouse, Inc., 3660 Waialae Avenue, Suite 304, Honolulu, Hawaii 96816-3236.

Limited 90-Day Warranty

Limited Warranty: SWH warrants to you, the original licensee, for your benefit alone, for a period of 90 days (the "Warranty Period"):

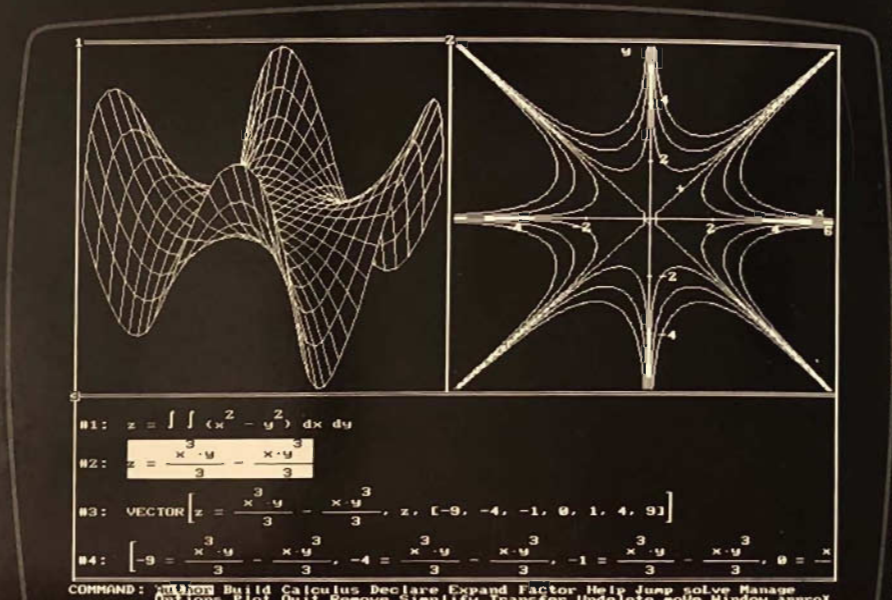
1. That SWH will replace the enclosed Disk and/or Manual, if physically defective, provided that (a) you did not damage the Disk and/or Manual, (b) the Disk has not been used in defective or incompatible equipment, (c) the Product Registration Card has been sent to SWH, and (d) the defective Disk and/or Manual is/are returned to SWH during the Warranty Period with proof of payment.

2. That the Software substantially conforms to the functional description in the Manual, or SWH will either correct the problem without charge to you or will refund your license fee, provided that during the Warranty Period you return the enclosed Disk to SWH together with proof of payment and a written description of the problem.

EXCEPT AS PROVIDED ABOVE, SWH MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SWH WILL NOT IN ANY EVENT BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL, OR EXEMPLARY DAMAGES, LOST DATA, OR LOST PROFITS, ARISING OUT OF USE OF OR INABILITY TO USE THE PRODUCT, EVEN IF SWH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY THIRD PARTY. SOME STATES OR COUNTRIES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, OR THE LIMITATION ON HOW LONG AN IMPLIED WARRANTY LASTS, SO SOME OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.


DERIVE
A Mathematical Assistant
PC compatibles
PC 9801 compatibles
Version 3.14
Copyright © 1988-1994 Soft Warehouse, Inc.
All rights reserved worldwide.
Soft Warehouse
HONOLULU • HAWAII

2000 Years of Mathematical Knowledge on a Disk


#1: z = ∫∫ (x^2 - y^2) dx dy
#2: z = x^3·y/3 - x·y^3/3
#3: VECTOR[z = x^3·y/3 - x·y^3/3, z, [-9, -4, -1, 0, 1, 4, 9]]
#4: [-9 = x^3·y/3 - x·y^3/3, -4 = x^3·y/3 - x·y^3/3, -1 = x^3·y/3 - x·y^3/3, 0 = x
COMMAND: Author Build Calculus Declare Expand Factor Help Jump solve Manage
Options Plot Quit Remove Simplify Transfer Undelete Move Window approX
Enter option
3D and contour plots of a double integral Free:97% Derive Algebra

DERIVE is a powerful computer algebra system for doing symbolic and numeric mathematics on your personal computer. *DERIVE*:

- Performs numerical operations exactly with no round-off error
- Approximates irrational expressions to thousands of digits of precision
- Algebraically simplifies, expands, and factors expressions; and solves equations
- Applies the rules of trigonometry, calculus, matrix algebra, and vector calculus
- Plots explicitly and implicitly defined functions in 2D with zooming and auto-scaling
- Generates 3D wire-frame function plots using hidden-line removal
- Displays and prints expressions using standard 2D mathematical notation
- Provides an easy to use, menu-driven interface with on-line help
- Is ideal for students, teachers, engineers, scientists, and mathematicians

System requirements:

- *DERIVE* (regular memory version): An IBM PC or compatible running MS-DOS version 2.1 or later with 512 kilobytes of memory and a 3½ inch (720K) diskette drive.
- *DERIVE* (Japanese version): An NEC PC-9801 or compatible running MS-DOS version 2.1 or later with 512 kilobytes of memory and a 3½ inch (1.2M) or a 5¼ inch (1.2M) diskette drive.
- *DERIVE XM* (extended memory version): A 386, 486 or Pentium™ based PC compatible running MS-DOS version 3.0 or later with 2 megabytes of extended memory and a 3½ inch (1.4M) diskette drive.

Printed on recycled paper with soy ink.

A Mathematical Assistant for Your Personal Computer

DERIVE®